On Quantum Programming Languages

Benoît Valiron

université
PARIS-SACLAY

On Quantum Programming Languages

Habilitation à diriger des recherches
de l'Université Paris-Saclay

présentée et soutenue à Gif-sur-Yvette,
le 24 Septembre 2024, par

Benoît VALIRON

Composition du jury

Pascale LE GALL Professeure des Universités, CentraleSupélec	Présidente
Luís SOARES BARBOSA Full Professor, Universidade do Minho	Rapporteur
Chris HEUNEN Professor, University of Edinburgh	Rapporteur
Emmanuel JEANDEL Professeur des Universités, Université de Lorraine	Rapporteur
Cyril BRANCIARD Chargé de Recherche CNRS, Institut Neel de Grenoble	Examinateur
Delia KESNER Professeure des Universités, Université Paris Cité	Examinatrice
Sophie LAPLANTE Professeure des Universités, Université Paris Cité	Examinatrice

Parrain

Pablo ARRIGHI
Directeur de Recherche, **INRIA**

université
PARIS-SACLAY

Titre : De l'étude des langages de programmations quantiques

Résumé : Cette thèse présente mes contributions à la recherche depuis la soutenance de mon doctorat en 2008. J'ai eu la chance de participer au développement des langages de programmation quantique depuis le début : le document a pour but de présenter mon point de vue sur l'évolution du sujet, mes contributions, et les tendances actuelles dans la communauté. Le document est destiné à un doctorant spécialisé dans les méthodes formelles.

Depuis 2008, la programmation quantique a énormément évolué. Alors que le matériel est passé d'expériences en laboratoire à des coprocesseurs quantiques prêts à l'emploi, les langages quantique sont passés de principes mathématiques abstraits à des environnements de développement complet. Leur conception doit désormais tenir compte du matériel ainsi que des cas d'usage. En outre, de nouveaux paradigmes de calcul non standard émergent, basés sur la possibilité de considérer la superposition des exécutions en plus de la superposition des données. Tout cela soulève des défis passionnants pour l'avenir.

La présentation commence par discuter l'état du domaine en 2008, avec une discussion sur les différentes approches telle que le lambda-calcul quantique—l'une de mes contributions avant 2008. Le manuscrit se penche ensuite sur trois sujets pour lesquels j'ai été activement impliqué.

La thèse se concentre d'abord sur l'avènement de langages de programmation quantique évolutifs. Dans ce contexte, j'ai participé au développement de Quipper, un langage de description de circuits inscrit dans Haskell, et de Qbricks, un outil de vérification déductive de programmes quantiques. Le deuxième thème principal de la thèse est la sémantique des lambda-calculs quantique : mes contributions concernent la description de sémantiques inspirées par des modèles de logique linéaire. La troisième partie de la thèse analyse la superposition d'exécutions de programmes : mes contributions sont le SWITCH quantique, montrant comment ce modèle ne peut pas être réduit à des circuits quantiques, et la conception de langages pour des programmes en superposition.

La thèse se termine par quelques tendances actuelles de la recherche : l'essor des langages graphiques, l'unification des contrôles quantiques et classiques, le développement de chaînes d'outils de compilation quantique, et l'analyse statique pour les programmes quantiques.

Title: On Quantum Programming Languages

Abstract: This thesis —*Habilitation à diriger des recherches*— presents some of my research contributions since my Ph.D defense in 2008. I have had the chance to participate in the development of quantum programming languages since their early developments: the presentation aims to present my point of view on the evolution of the subject, my contributions, and the current research trends in the community. The target audience is a graduate student interested in pointers to the field of quantum programming languages

Since 2008, the state of quantum programming has evolved tremendously. As quantum hardware moved from physical artifacts in bench labs to ready-to-use quantum coprocessors, quantum programming languages evolved from abstract mathematical principles to scalable proposals. Language designs now need to consider constraints coming from both hardware and use cases. Furthermore, novel, non-standard computational paradigms are emerging based on the possibility of considering the superposition of executions on top of the superposition of data. All of this raises exciting challenges for the years to come.

The presentation starts with a discussion of the state of the field of quantum programming language in 2008, with a discussion on the attempts at toy languages, and in particular, the quantum lambda calculus—one of my contributions before 2008. The manuscript then dives into three main subjects I have been actively involved in.

The thesis first focuses on the advent of scalable quantum programming languages. In this context, I have participated in developing Quipper, a domain-specific, circuit-description language embedded in Haskell, and Qbricks, a tool for deductive verification of quantum programs. The second main topic of the thesis consists of the semantics of quantum lambda-calculi: My contributions concern the description of semantics inspired by models of linear logic. The third part of the thesis analyses the superposition of program executions: my contributions are the quantum SWITCH, showing how this model cannot be reduced to quantum circuits, and the design of languages for programs in superposition.

The thesis concludes with a few current research trends: the rise of graphical languages, the reconciliation of quantum and classical control, the development of quantum compilation toolchains, and static analysis for quantum programs.

Remerciements

Je tiens tout d'abord à remercier chaleureusement Luis Soares Barbosa, Chris Heunen et Emmanuel Jeandel d'avoir accepté de rapporter ce manuscrit, ainsi que Cyril Branciard, Delia Kesner, Sophie Laplante et Pascale Le Gall pour leur présence sur le jury. Je remercie finalement Pablo Arrighi pour son role de parrain qui fut décisif dans la finalisation de ce manuscrit.

Cette habilitation est une histoire qui a commencée en 2018, quand Frédéric Boulanger a accepté de donner sa casquette HDR pour l'encadrement de 이동호 (Dongho Lee). Il était entendu que je passerais mon HDR d'ici à la soutenance. Cela n'a pas vraiment été le cas, et j'ai eu recours de nombreuses fois à Frédéric, mais aussi Marc Baboulin, Gilles Dowek et Pablo Arrighi pour servir de co-encadrants. Je tiens à les remercier ici pour leur aide.

Grâce à leur soutien, j'ai eu l'honneur d'accompagner jusqu'à leur soutenance Timothée Goubault de Brugière, 이동호 (Dongho Lee), Kostia Chardonnet, Agustin Borgna et Louis Lemonnier. À l'heure où j'écris ces lignes, Jérome Ricciardi, Nicolas Heurtel et Julien Lamiroy sont toujours en thèse. J'espère pouvoir assister à leur soutenance en ayant finalement terminer cette habilitation qui est devenue au fil du temps l'arlésienne de l'équipe. Je remercie tous mes étudiants, qu'ils aient fini ou non: l'interaction avec eux forment le cœur de mon attachement au travail de recherche.

Je remercie aussi les collègues de l'équipe QuaCS qui a vu le jour pendant la (longue) gestation de cette habilitation. L'arlésienne qu'est devenue cette habilitation a généré un fond de blagues qui ont maintenu une pression salutaire.

En dernier lieu, je remercie ma famille pour leur compréhension et leur soutien, pendant tous ces soirs et week-ends durant lesquels j'ai été particulièrement peu disponible.

Pour Caroline, sans qui la vie serait moins belle.

Contents

Chapter A

Introduction

The birthdate of quantum computing can be traced back to 1982 when Feynman [Fey82] envisioned using controllable quantum systems to simulate physical behavior. Since then, quantum computing has grown into a thriving research field, with foreseen applications in a wide range of topics: quantum simulation for pharmaceuticals and chemicals applications, quantum linear algebra for AI and machine learning, quantum optimization and search, quantum factorization, *etc*. As of December 2022, more than 430 algorithms were registered on the Quantum Algorithm Zoo [QZOO22], a comprehensive catalog of quantum algorithms.

In parallel to this boom in algorithm design, the investment in quantum computing is reaching an all-time high, both from public and private actors. France set up the "*Plan Quantique*" last year, while Europe launched the "Quantum Flagship" a few years ago. In the previous five years, both the number of startups and the investment in quantum computation have skyrocketed to more than 200 startups worldwide, and 25 billion dollars of public investment as of 2021 [McKinsey21].

The idea behind quantum computing is to code information on the state of objects governed by the laws of quantum physics. The mathematical theory is well established [NC02] and makes it possible to reason on what is doable—and what is not — in this computation paradigm without having to rely on concrete hardware. The state of a quantum memory can be regarded as a complex, linear combination of bit-strings. This mathematical formalism entails two of the main features of quantum computation: superposition of data and entanglement. On the downside, reading information from a quantum memory is a destructive and probabilistic operation, modifying the global state of the memory.

One of the first and maybe most notorious quantum algorithms, Shor's factoring algorithm [Sho94], placed quantum computation on the radar for potentially disruptive technologies. However, in part due to the lack of scalable hardware at the time, later quantum algorithms were typically developed as tools to explore the complexity speedup entailed by using a quantum memory. If this purely theoretical approach to quantum computation uncovered interesting foundational results, it remained somehow far from concrete instantiations.

Quantum algorithms such as the one devised by Shor require a level of abstraction higher than what can be needed by Feynman's vision of quantum simulation. Knill [Kni96] proposed in 1996 a rationale for what a quantum coprocessor should permit to implement such algorithms. This analysis is the basis of a wide range of works on the *computer science* aspect of quantum computation. On one end of the spectrum, a series of research developments discuss concrete quantum programming languages or libraries for interacting with a quantum memory [BCS03, Öme00, Öme03]. At the other end of the spectrum, Knill's description seeded lines of research on the *semantics* of quantum programming language based on models of linear logic, domain theory and category theory [Sel04a, Sel04b, Ton04, Val04, Val08], the latter cross-fertilizing with early formalizations of quantum information and quantum protocols

[Coe04, AC04].

The 2010s have seen a rapid upscaling of the proposals pertaining to quantum programming. Stirred by the active development of hardware and integrated quantum coprocessors, the field has moved from blackboards and lab benches to industrial use cases. Although the hardware is still in the so-called *Noisy Intermediate-Scale Quantum* (*NISQ*) state with quantum memory of a few hundred uncorrected qubits subject to decoherence [Pre18], the research in quantum programming and compilation is thriving in proposals for making use of these memories [BCKH+22]. Indeed, the memories are large enough to make it impractible to manipulate by hand, yet constrained enough to require dedicated languages and sophisticated compilation and optimization techniques. The field is furthermore opening the path toward *Large Scale Quantum* (*LSQ*) computation, for when the hardware will be able to support error correction.

The arrival of concrete machines with more than a handful of qubits has created a strong pull effect for the development of quantum programming languages [QTOOLS24]. Indeed, if hand-writing circuits is feasable for a few qubits, it quickly becomes cumbersome and error-prone even for a few dozen of qubits. Turning a pen-and-paper algorithm to an effective, runnable sequence of gates on a quantum coprocessor with a hundred qubits requires a programming language to describe and manipulate the structure of the circuit. Whether this language is standalone or embedded, it has to be formalized enough to support analysis techniques and tools to assess the validity of quantum programs. Moreover, whether in the NISQ or the LSQ era, quantum coprocessors are and will arguably be both expensive to run and limited in resources. Optimizing the resource footprint of an algorithm is then critical, opening the field to the development of synthesis and optimization techniques for quantum programs.

> I entered the game in 2002 when I started a Master's program at the University of Ottawa under the supervision of Peter Selinger. I had the opportunity to design a quantum lambda calculus and work on its semantics, first as a Master's student and then as a PhD student, still under Peter Selinger's supervision. I had, therefore, the chance to be on the frontline of the research on quantum programming languages.
> Since my Ph.D. defense in 2008, the field has evolved and significantly matured, and I was lucky to be part of the process. The rest of the manuscript summarizes this evolution, seen from my own view, and focuses on my contributions until now. Following the rules of the *Habilitation à diriger des Recherche (HDR)*, I take the year 2008—time of my Ph.D. defense— as a pivot and focus on what happened next. The notion of "present time" being a moving target, I chose 2020 (give or take) as the end of the past and the beginning of the future—this is the time where I started to write this thesis.
> I shall use blue text boxes like the one encapsulating this paragraph to reflect on my experience with the topics and subjects discussed.

Scope and plan of the Manuscript. This thesis is concerned with quantum programming languages from a formal perspective. The scope is deliberately skewed towards the research interests—and the work—of the author. The document focuses on three emblematic research threads (Chapters C, D and E) in which the author participated between 2008 and 2020. For each chapter, the related publications are summarized at the end in a separated table: Tables D.8, C.13 and E.1. The later publications are in Table F.1.

Chapter B briefly presents some background material and reviews the state of quantum programming languages as it was in 2008. The discussion covers the preliminary design proposals for quantum languages with a focus on the quantum lambda calculus, one of our contributions at the time.

Chapter C discusses the design of quantum programming languages within the coprocessor model and the shift from toy languages to scalable programming environments. We have, in particular, been involved in the development of Quipper, a domain-specific, circuit-

description language embedded in Haskell. The language comes with sound design principles that are still state of the art nowadays. We then discuss two related aspects: circuit synthesis and quantum program certification. Our contributions to circuit synthesis are twofold: a strategy for generating oracles by turning a classical piece of code into a family of reversible circuits and approaches using numerical analysis to automate circuit synthesis based on a matrix-like description. Regarding program verification, we have been involved in deductive verification, particularly with developing the tool Qbricks.

Chapter D explores the semantics of quantum lambda calculus and its extension with circuit-description capabilities, bridging with Chapter C. A semantics is a formal description of some of the properties of a programming language: its structures, its behavior, its action, etc. A semantics usually shares a strong connection with a model of some logic through a Curry-Howard correspondence. In the case of quantum computation, one of our contributions before 2008 has been to connect quantum lambda calculus with linear logic. This resource-sensitive logic forms a natural framework for reasoning on the non-duplicability of quantum registers. The chapter presents such Curry-Howard correspondence, linking linear logic and the quantum lambda calculus. Three aspects of the problem are then discussed: first, the quest for a denotational semantics for the corresponding typed quantum lambda calculus; then the link between the typed quantum lambda calculus and an interaction-based model of linear logic: Geometry of Interaction; finally, the description of denotational semantics for quantum lambda calculi with circuit-description features.

Chapter E examines an effect specific to quantum computation: quantum control. This non-standard model of computation generalizes the notion of superposition: On top of the superposition of data, we consider the possibility of superposition of executions. All of this opens several challenges, such as the expressivity of this new computational paradigm and the design of syntactic languages to describe superpositions of programs. Concerning the expressivity of the model, our seminal contribution is the quantum Switch: a minimal algorithm featuring a superposition of execution that quantum circuits cannot represent. On the syntactic aspect, we explored several approaches for functional languages with superposition of programs, using not only lambda calculus but also specific syntax based on pattern-matching.

Finally, Chapter F describes a few research trends in the community corresponding to the interest of the author: the rise of graphical languages, the problem of the unification of quantum and classical control, the definition of quantum compilation toolchains, and the challenge of quantum program certification.

Blind Spot. Many important aspects pertaining to the field of quantum languages have been left undiscussed in this document. In particular, we shall not consider the problem of error correction, the development of quantum algorithms, or (most) practical aspects of quantum compilation. Graphical languages and ZX in particular will not be discussed to the extent they deserve—again, the scope of the document is only the author's existing research production. It should also be understood that we shall not discuss industrial-scale quantum programming languages; the presentation stays at a formal, theoretical level.

Audience. The target audience for this document is a graduate student or a researcher in formal methods interested in acquiring pointers to the field of quantum programming languages. More precisely, the main techniques used in the various works described in the document are rewriting, logical and type systems, and category theory. The reader should not expect complete proofs of results; instead, the presentation tries to stay high-level, and the main results and constructions are, in general, given through the use of examples.

Chapter B

Quantum Programming back in 2008

Since 2008, the state of the art in quantum programming languages has evolved a lot, particularly with new actors shaping the field in inconceivable ways back then. This chapter is devoted to summarizing the state of the field back then.

- Section B.1 offers a quick introduction to quantum computation in general. On top of the mathematical foundations, the section describes the standard computational paradigm for quantum computation: the quantum coprocessor model. The section also discusses an emerging language at the time with a graphical notation: the ZX calculus. This section sets the playground for the rest of the thesis.
- Section B.2 presents the state of quantum programming languages around 2008. The section discusses a few emblematic approaches and design strategies. Indeed, some of them are at the root of the subsequent developments. For instance, in hindsight, the notion of circuit-description language was already there in the quantum IO monad of Altenkirch&Green's [AG09], most of the challenges about quantum control were established in for QML [AG05a], and Bettelli and Ömer had already started pondering the interaction with the quantum coprocessor [BCS03, Öme03].
- Section B.3 introduces one of our main contributions at the time: the quantum lambda calculus [Val04, Val08, SV06]. This extension of the lambda calculus provides a theoretical framework for reasoning on functional programming languages accommodating quantum computation. The section discusses the necessity for a linear type system due to the non-duplicability of quantum information and sketches the existing denotational semantics at the time. As the section shows, existing semantics for the quantum lambda calculus were still very modest. In particular, they could not capture both duplicable and non-duplicable data at the same time [SV08a, SV08b].

B.1 Primer on Quantum Computation

In this section, we lay out the background on quantum computation needed for the rest of the thesis. We invite the reader to consult e.g. [NC02] for more details.

B.1.1 Quantum Memory

In the standard model of quantum computation, one has access to a quantum coprocessor holding a special kind of *quantum* memory. It consists of data encoded on the state of quantum

particles: photons, ions, *etc.* The behavior of the quantum memory is derived from the rules of quantum mechanics. From a mathematical standpoint, the *state* of the quantum memory is a normalized vector in a Hilbert space [NC02], usually considered modulo a global phase —i.e. multiplication by a (non-zero) scalar. The smallest piece of quantum information is the *quantum bit*, or *qubit*: its state is represented by a normalized vector in the Hilbert space Q of dimension 2. One chooses a basis $\{|0\rangle, |1\rangle\}$ of two orthonormal vectors: a canonical representation for the state $|\phi\rangle$ of a qubit is therefore of the general form

$$|\phi\rangle = \rho_0 |0\rangle + \rho_1 e^{i\theta} |1\rangle,$$

where ρ_0 and ρ_1 are positive reals such that $\rho_0^2 + \rho_1^2 = 1$ and θ is an angle between 0 and 2π. The value θ is a *phase*, whereas ρ_0 and ρ_1 are called *amplitudes*. In general, we however simply consider a representative $\alpha |0\rangle + \beta |1\rangle$ with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$, keeping in mind that the *global phase* $\frac{\alpha}{|\alpha|^2}$ is not relevant.

The notation $|\phi\rangle$ is called a *ket*: if we choose the lexicographic ordering on the basis $|0\rangle, |1\rangle$, then $|\phi\rangle$ stands for the column vector

$$|\phi\rangle = \alpha |0\rangle + \beta |1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

There is a dual notation for row-vectors —or functionals—, the *bra*. The conjugate transpose of $|\phi\rangle$ is therefore

$$\langle\phi| = \begin{pmatrix} \bar{\alpha} & \bar{\beta} \end{pmatrix}.$$

The notation is consistent with the *scalar product*, as follows:

$$\langle\phi \mid \phi\rangle = (\langle\phi|)(|\phi\rangle) = \begin{pmatrix} \bar{\alpha} & \bar{\beta} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = |\alpha|^2 + |\beta|^2 = 1.$$

The basis $|0\rangle, |1\rangle$ therefore forms an *orthonormal basis*. Another standard orthonormal basis is $|+\rangle, |-\rangle$, where $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

Kronecker product When considering several quantum registers simultaneously, the state of the overall system lies within the *Kronecker product*, or *tensor product* of the individual state spaces. If Kronecker products can be defined through a universal property [Lan02, Ch. XVI], a simple presentation can be done with spaces equipped with bases. Consider the two Hilbert spaces $\mathcal{E}$ and $\mathcal{F}$ of respective bases $B_{\mathcal{E}} = \{e_i\}_i$ and $B_{\mathcal{F}}\{f_j\}_j$. The space $\mathcal{E} \otimes \mathcal{F}$ is defined as the Hilbert space with (formal) basis $B_{\mathcal{E}} \times B_{\mathcal{F}} = \{(e_i, f_j)\}_{i,j}$. We write the pair (e_i, f_j) as $e_i \otimes f_j$, and we bi-linearly extend the notation $- \otimes -$ to linear combinations as follows:

$$\left(\sum_i \alpha_i \cdot e_i\right) \otimes \left(\sum_j \beta_j \cdot f_j\right) = \sum_{i,j} (\alpha_i \beta_j) \cdot (e_i \otimes f_j).$$

The ket- and bra-notations make it possible to shorten the tensor notation: we write $|00\rangle$ for $|0\rangle \otimes |0\rangle$. The canonical basis for a 2-qubit system is then in lexicographic ordering $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$. Unless stated otherwise, the convention is to always use the lexicographic ordering for basis.

Consider two registers A and B of respective states $|\phi\rangle_A \in \mathcal{H}_A$ and $|\psi\rangle_B \in \mathcal{H}_B$. The state of the joint system AB is then $|\phi\rangle_A \otimes |\psi\rangle_B \in \mathcal{H}_{AB} = \mathcal{H}_A \otimes \mathcal{H}_B$. Such a state is called *separable*. Every state is however not necessarily separable: suppose for instance that A and B are both single qubits. A valid state for the 2-qubit system AB is

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

which cannot be factorized as $|\phi\rangle \otimes |\psi\rangle$: such a state is called *entangled*.

$$H \triangleq \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \qquad R_Y(\theta) \triangleq \begin{pmatrix}\cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2})\\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2})\end{pmatrix}$$
$$X, NOT \triangleq \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} \qquad R_X(\theta) \triangleq \begin{pmatrix}\cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2})\\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2})\end{pmatrix}$$
$$Z \triangleq \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} \qquad R_Z(\theta) \triangleq \begin{pmatrix}e^{-i\frac{\theta}{2}} & 0\\ 0 & e^{i\frac{\theta}{2}}\end{pmatrix}$$
$$S \triangleq \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix} \qquad T \triangleq \begin{pmatrix}1 & 0\\ 0 & e^{\frac{i\pi}{4}}\end{pmatrix}$$

Table B.1: Examples of 1-qubit Quantum Gates

B.1.2 Quantum Operations

The operations one can perform on a quantum memory are of two kinds: *unitary operations* and *measurements*. The former correspond to actions internal to the quantum memory, without feedback, while the latter models classical information retrieval from the quantum memory.

Unitary Operations. A unitary operation corresponds to a unitary endomorphism on the space of states of the quantum memory —as in linear algebra—. In particular, such an operation is linear, invertible, and sends orthonormal bases to orthonormal bases.

In general, a quantum coprocessor only supports a small set of unitary operations, called *unitary gates*, or *quantum gates*. They usually only act on one or two qubits at a time. A standard list of such gates acting on one qubit can be found in Table B.1. Standard gates acting on two qubits are the control-NOT and the SWAP gate

$$CNOT = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}, \qquad SWAP = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix},$$

CNOT sends $|0x\rangle$ to $|0x\rangle$ and $|1\rangle \otimes |x\rangle$ to $|1\rangle \otimes |\neg x\rangle$: it leaves invariant the subspace $|0\rangle \otimes Q$ and acts as X on the second qubit in the subspace $|1\rangle \otimes Q$. The SWAP-gate is sending $|xy\rangle$ to $|yx\rangle$: Note how we use the lexicographic ordering on bases to represent the matrices.

In general, given a unitary U acting on n qubits, the *control* of U is the gate $C-U$ acting on $Q \otimes Q^{\otimes n}$ and defined as $|0\rangle \otimes |\phi\rangle \mapsto |0\rangle \otimes |\phi\rangle$ and $|1\rangle \otimes |\phi\rangle \mapsto |1\rangle \otimes (U|\phi\rangle)$. Using the lexicographic representation of basis states, the gate $C-U$ can be represented as the block-matrix

$$C-U = \begin{pmatrix}1 & 0\\ 0 & U\end{pmatrix}.$$

As an example, the gate *CNOT* is $C-X$. We can also define the *Toffoli gate* that acts on 3 qubits and which is defined as $C-CNOT$. It sends $|xy\rangle \otimes |z\rangle$ to $|xy\rangle \otimes |z \oplus (x \wedge y)\rangle$.

A quantum gate-set is suitable for general quantum computation if it is *universal*, i.e. if any unitary map acting on n qubits can be realized with composition and tensors of elementary gates. Depending on the gate-set, this realization can be exact, or approximate up to an arbitrary small error.

Measurement. A measurement corresponds to the (classical) observation of a quantum system whose state lives in the Hilbert space $\mathcal{H}$ to retrieve a classical piece of information. Operationally, it consists in choosing two orthogonal subspaces $\mathcal{H}_0$ and $\mathcal{H}_1$ spanning $\mathcal{H}$ and determining whether the state of the system belongs to $\mathcal{H}_0$ or $\mathcal{H}_1$. In this case, any vector $|\phi\rangle \in \mathcal{H}$ can be decomposed in $|\phi\rangle = \alpha|\phi_0\rangle + \beta|\phi_1\rangle$, with $|\phi_0\rangle \in \mathcal{H}_0$ and $|\phi_1\rangle \in \mathcal{H}_1$, and $|\alpha|^2 + |\beta|^2 =$

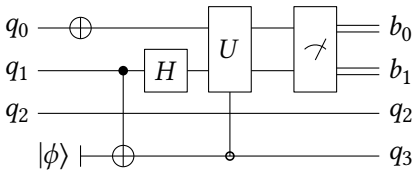


Figure B.2: Example of quantum circuit

1. A measurement against the decomposition $\mathcal{H} = \mathcal{H}_0 \oplus \mathcal{H}_1$ will project $|\phi\rangle$ onto one of the two subspaces with some probability: $|\phi\rangle$ is changed to $|\phi\rangle_0$ or $|\phi\rangle_1$ (modulo renormalization) with probability respectively $|\alpha|^2$ or $|\beta|^2$. The *classical result* of the measurement is the subspace to which the state now belongs.

For instance, measuring a qubit $\alpha|0\rangle + \beta|1\rangle$ along the basis $\{|0\rangle, |1\rangle\}$ —that is, the decomposition $\mathcal{Q} = (\mathbb{C}|0\rangle) \oplus (\mathbb{C}|1\rangle)$— turns the qubit into $|0\rangle$ with probability $|\alpha|^2$, in which case we get the classical result "0", or into $|1\rangle$ with probability $|\beta|^2$, in which case we get the classical result "1". By convention, the result "1" stands for "True" and "0" for "False".

Quantum Circuits. Unitary gates are used to realize a global unitary operation on the memory state-space. Such operations are historically represented using the ad-hoc, graphical notation of *quantum circuit* [Yao93]. Quantum circuits form the quantum counterpart of classical, Boolean circuits. Due to the peculiar nature of quantum data, they are however much simpler than Boolean circuits: there is no branching nor possible loop-back. Simple horizontal lines read from left to right represents the life-span of a qubit or a quantum register, and boxes on wires represents operations on them. An example is shown in Figure B.2. Wires can be labeled. In the example circuit, q_0, q_1 and q_2 are input wires of the circuit, while q_3 is initialized with $|\phi\rangle$. The NOT-gate is represented with a $\oplus$-symbol, and generic boxes are rectangles: H acts on q_1, while U acts on q_0 and q_1. U is negatively controlled by q_3, while the NOT-gate on q_3 is positively controlled by q_1. Circuits can also contains measurements, shown in Figure B.2 as the last box on the right. Boolean results are represented with double-wires and can be labeled for easy referring.

Quantum circuits can be extended with more constructs: measurements, wire initialization (such as shown in Figure B.2), wire termination, *etc*. In a quantum circuit wires that are initialized and then terminated inside the circuit correspond to temporary registers. They are called *auxiliary* wires, or *ancillas*. Ancillas are more subtle to use than conventional, local variables: terminating an ancillas amounts to measure the corresponding qubit. If needed, special care must therefore be taken to retain unitarity.

Hardware The mathematical model is an ideal representation of the memory setup at the hardware level. Indeed, physical qubits are noisy, as they are subject to decoherence [Sch08]. The hardware also entails topological constraints —it might not be possible to act on two physically distant qubits— or limitations on the gate set. From a programming perspective, these problems are to be addressed in the context of a quantum compilation toolchain.

B.1.3 Mixed States

An arbitrary sequence of operations sent to a quantum memory interleaves unitaries and measurement. In general, the resulting state of the quantum memory at the end of the computation is therefore not a single quantum state $|\phi\rangle$ —a *pure state*— but a *mixed state* —whose naive representation would be a probability distribution of pure states. Does this mean that one can use the set of such probability distributions as a valid model for mixed states? The answer is not so clear: it depends on what *observations* are allowed.

Superoperators. If a quantum computation is understood as a quantum experiment, the only available operations are unitaries and measurements, and the only possible classical outcome of an observation is the (classical) result of a measurement (did we measure 0 or 1?). In this configuration, as physicists already noticed [NC02] probability distributions do not make a sound model for mixed states. Instead of considering $\sum_i p_i\{|\phi_i\rangle\}$, a semantics matching the observational equivalence given by unitaries and measurements is the *positive matrix*

$$\rho = \sum_i p_i \, |\phi_i\rangle \, \langle\phi_i|$$

A positive matrix of trace 1 (such as this one) is also called a *density matrix*. Positive matrices form a semantics for mixed states supporting both unitary operations and measurements. In this framework, a general quantum computation inputting n qubits and outputting m quantum bits is represented by a trace-preserving, *completely positive map* —also known as *superoperator*—

$$F : \mathbb{C}^{2^n \times 2^n} \to \mathbb{C}^{2^m \times 2^m}. \tag{B.1}$$

A completely positive map (CPM) is a linear map such that for all $k \in \mathbb{N}$, $F \otimes \mathrm{id}_{\mathbb{C}^{k\times k}}$ sends positive matrices to positive matrices.

Löwner order. Positive matrices (and by extension completely positive maps) feature an ordering relation, the *Löwner order* [Löw34, Loe50, BB83]: $A \sqsubseteq B$ whenever $B - A$ is positive. This order is stable under sum and (non-negative) scalar multiple. It makes the cone of positive $n\times n$ matrices a *bounded dcpo*: any bounded, directed set of positive matrices under the Löwner order admits a least upper bound. This relation is consistent with the trace: if $A \sqsubseteq B$ then $\mathrm{tr}(A) \leq \mathrm{tr}(B)$.

The Löwner order makes it possible to interpret possibly non-terminating quantum programs using positive matrices of trace less or equal to 1 and trace *non-increasing* completely positive maps [Sel04a]. Following the standard domain interpretation [Plo83], the 0-valued element —bottom of the cone— corresponds to the diverging program.

B.1.4 Quantum Coprocessor Model

In order to move from a mathematical model based on Hilbert spaces—or from physics experiments—to a programmable model of computation, Knill proposes in 1996 the *QRAM model* [Kni96], with a few basic pseudo-code constructs to express quantum algorithms.

Although other computational paradigms exist such as Measurement-based computation [RBB03, RB01] or adiabatic computation [FGGS00, FGGLLP01], Knill's QRAM model has so far remained the standard model used in the design of quantum algorithms [QZOO22]. Indeed, from the perspective of the quantum coprocessor, the run of a quantum algorithm can be summarized with three classes of operations: quantum register initializations; application of elementary gates on arbitrary qubits; measurements of arbitrary qubits.

Knill's QRAM model requires these low-level operations since they form the core of the interactions between the classical machine and the quantum coprocessor. In addition to these elementary building blocks, Knill proposes a few high-level constructs such as subroutine inversion or controlling.

He also proposes to support the invocation of classical operations as oracles. An *oracle* is the translation of the classical structure of the program into a quantum unitary. It can for instance be the structure of a graph, an arithmetic operation, *etc*. In general, provided that the description of the problem is written as a (classical, irreversible) map $f : \mathtt{bool}^n \to \mathtt{bool}^m$, one can always build the quantum unitary $U_f : \mathcal{Q}^{\otimes n} \otimes \mathcal{Q}^{\otimes m} \to \mathcal{Q}^{\otimes n} \otimes \mathcal{Q}^{\otimes m}$ shown in Figure B.3 and defined as

$$U_f : |x\rangle \otimes |y\rangle \mapsto |x\rangle \otimes |y \oplus f(y)\rangle \, .$$

Oracle generation is the topic of Section C.2.1.

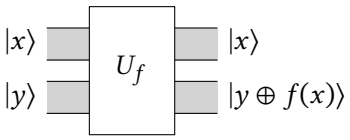


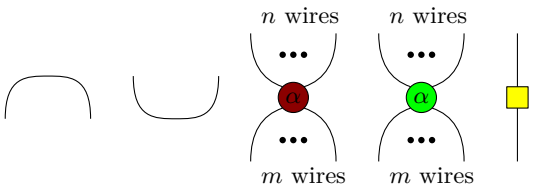
Figure B.3: Structure of the typical oracle

B.1.5 ZX calculus

Nowadays, graphical calculi for quantum computation are commonplace [CK17]. However, if physicists were already making good use of graphical representations with e.g. Feynman diagrams [Wüt11], in 2008 there were seldom graphical languages for quantum computation.

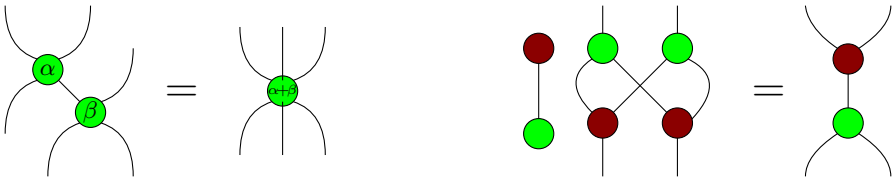
Apart from the ad-hoc representation of quantum circuits, in the late 2000's a steady trend was however taking off. Building on category theory and led by Abramski and Coecke [Coe04, AC04], the computer science research group at Oxford became a thriving center for a novel graphical representation based on the interaction of pairs of mutually unbiased observables: the ZX calculus [CD07, CD08, pub22, CK17].

Falling within the large class of tensor network representation [BB17], the ZX calculus can be regarded as a graphical language for a special kind of dagger, compact closed category with two commutative †-Frobenius monoids [CPV13]:



(B.2)

ZX-diagrams are composed from these constructs and read from top to bottom. The *green spider* corresponds to the basis $\{|0\rangle, |1\rangle\}$ and the *red spider* to the basis $\{|+\rangle, |-\rangle\}$. The α is a phase. These two algebras form a bialgebra satisfying the Hopf law [CD08, CD11], so for instance



where a node with no label corresponds to the phase 0.

The ZX calculus aims at abstracting away the structure of finite Hilbert spaces. A ZX term indeed admits a standard representation as general, linear function acting on Hilbert spaces. A diagram with n input and m output wires corresponds to a linear function $Q^{\otimes n} \to Q^{\otimes m}$. For instance, the "cap" in Eq. (B.2) is the map $\mathbb{C} \to Q \otimes Q$ sending 1 to $|00\rangle + |11\rangle$, while the green node is sending $|1 \cdots 1\rangle$ to $e^{i\alpha} |1 \cdots 1\rangle$ and any other basis state to itself.

The ZX calculus is mentioned later in Section C.2.3.

B.2 Quantum Programming Paradigms

Before 2010, with the lack of concrete quantum coprocessors and use-cases for quantum algorithms, quantum programming was mostly a theoretical playground [Gay06]. Nonetheless, with hindsight, much of the current state-of-the-art techniques in quantum programming

were already latent. We present them in this section, building on five works spanning the approaches at the time: Bettelli's C++ library [BCS03], Ömer's QCL language [Öme03], Altenkirch&Grattage's QML for quantum control [AG05a], Altenkirch&Green's quantum IO monad [AG09], Vizzotto's quantum arrows [VAS06]. An important paradigm for this thesis is the quantum lambda calculus [SV06]: we discuss it in Section B.3.

Quantum Programming within Classical Environment. A quantum algorithm aims at solving a classical problem instance, and it is meant to run on a classical computer, piloting a quantum coprocessor. As such, the *control flow* of the program is purely classical. It therefore makes sense to package the interaction with the quantum coprocessor into a dedicated library of an existing programming language. Bettelli *et al* [BCS03] proposes such a library within C++, capitalizing on C++ object model to build circuit abstraction. Although this particular implementation has not spurred any spin offs, the concept of using an existing (classical) programming language to host a quantum language has been very successful and is still in use in most current programming environment for quantum computation such as Qiskit. The limit of this approach is however the ability to reason about quantum programs and to offer tools for certified quantum programming.

Circuits as Side-Effects. Instead of using C++ as a host language, Altenkirch&Green [AG09] proposes a Haskell *domain-specific language* (*DSL*), building on Haskell's monadic paradigm to abstract away the interaction with the quantum coprocessor. Altenkirch&Green [AG09] in fact presents the *first formal* baseline for a sound understanding of the interaction with the quantum coprocessor: it can be understood as an input/output side-effect. A quantum program *outputs* gates to the coprocessor, while it *inputs* results of measurements. Haskell makes it possible to give a clear interface to a side effect. With the *quantum IO* monad QIO, one can therefore type qubit initialization and measurements as

```
1 qinit :: Bool -> QIO Qbit
2 meas  :: Qbit -> QIO Bool
```

that is, `qinit` inputs a Boolean value and returns a qubit object within the QIO interface: such an operation only makes sense within the context described by the interface. An implementation can be a real quantum coprocessor, or a simulator, or some more exotic implementation for instance recording all possible execution traces. The quantum IO monad framework forms the baseline for the development of Quipper discussed in Section C.1.2.

Circuits as Functions. Unlike Bettelli's approach [BCS03] a quantum circuit in the QIO framework is a regular function in the host language. A circuit on one wire is typed with

```
Qbit -> Circ Qbit.
```

The input wire of the circuit is the input of the function, and the output wire of the circuit is the output of the function. As a side-effect, the function generates a piece of circuit. The `Circ` type constructor encapsulates all the interaction between the program and the coprocessor.

Using regular functions to model circuits might limit the amount of manipulation allowed on circuits. Some operations such inversion cannot be easily formulated in such a general context. An alternative proposal has been formulated by Vizzotto *et al* [VAS06]. The proposal builds on Haskell's implementation of *arrows*: a contrived notion of function, distinct from Haskell's usual function-type. This *quantum arrow* can therefore encapsulate all of the interaction with the quantum coprocessor, and it offers an alternative approach to the QIO monad—albeit arguably less intuitive to program. Nonetheless, the two layers of arrows (the special quantum arrow and the regular, Haskell arrow-type) are very versatile, and can be seen as a foundation for Theseus [JS14] and the contributions presented in Section E.3.2.

Programming Constructs of the Quantum Coprocessor. Although in the 2000's quantum coprocessors were still a very theoretical notion, there were already attempts at exploring their programming capabilities. Instead of simply stacking gates into a circuit, Ömer proposes with the imperative language QCL [Öme03] quantum-specific subroutines, making it possible to distinguish features only available classically or only available quantumly. QCL can in a sense be regarded as a preliminary exploration of the current trend of hybrid quantum programming.

The language QCL is very imperative and the approach fails to catch the control flow hidden *inside* a quantum circuit. A naïve *quantum control* consists in the usual control of unitary, seen as a *quantum test*: an operator U acting on wire q_0 and controlled by wire q_1 can be regarded as a test on q_1. The language QML [AG05a] is arguably the first one to offer such syntactic, purely quantum test. The authors derive a small first-order language in which such a control can be written with an `if-then-else` construct: the test is quantum in the sense that the value tested upon is never measured, and both branches of the test fire in superposition. Quantum control is however an elusive notion, and besides simple tests, allowing general superpositions of execution paths has been shown highly non-trivial and is still an active research area, discussed in Chapter E.

B.3 Quantum Lambda Calculus

One of the main topics of this thesis is the quantum lambda calculus [SV05, SV06, Val04, SV09, Val08]. This language formalizes the notion of quantum, higher-order functional programming language with classical control, where a program is running on a classical computer with access to a quantum coprocessor. The language is equipped with a set of constructs and an operational semantics to formalize the interaction with the coprocessor. This approach has been shaping what is now the state of the art in term of quantum programming and quantum program certification.

This section can be regarded as a quick introduction to Chapter D. We first briefly recall the lambda calculus. We then discuss the strategy employed in [SV06] to extend it to support quantum computation: the resulting formal language is the quantum lambda calculus, our main contribution before 2008. We then present the notion of type system we developed using linear logic, discussing why it makes a suitable framework for typing quantum data. We finally quickly sketch the state of denotational semantics of the quantum lambda calculus in 2008.

> This has been the subject of my M.Sc. [Val04] and my Ph.D. thesis [Val08]: my main contribution at the time has been the study of the quantum lambda calculus and of its semantics.

B.3.1 Lambda Calculus

The lambda calculus [Bar84] is a versatile model of higher-order programming languages, where functions are first-class terms that can be returned or passed along as arguments. The basic constructs consist of variables: $x, y, \ldots$, lambda-abstractions $\lambda x.M$, standing for functions of argument x and body M, and applications MN: the term N is an argument fed to the function M. Terms of the language are called *lambda-terms*. A variable x in a term M may be *bound* by a lambda; otherwise it is called *free*. Computation is typically defined with a rewrite-system based on the so-called *beta-reduction*:

$$(\lambda x.M)N \to M[x := N].$$

Various constraints can be set, yielding *evaluation strategies*: call-by-value, call-by-name, call-by-need, *etc* [Plo75, MOTW95], *etc*. The language can furthermore be extended with constants

and other constructs to natively support other programming features and/or side-effects.

Lambda-terms can be typed [BDS13]: the grammar of types consists at least of one constant type α and an arrow constructor $A \Rightarrow B$. A term $\lambda x.M$ being a function, its type is of the form $A \Rightarrow B$, when x is meant to be of type A and M of type B. A set of *typing rules* then specify what is a valid type for a given term. For instance, if M is of type $A \Rightarrow B$ and N is of type A, then MN can be specified of type B:

$$\frac{M : A \Rightarrow B \quad N : A}{MN : B}. \tag{B.3}$$

Typed lambda calculi form the canonical medium for the *Curry-Howard isomorphism*: a correspondence identifying types with logic formulas, and terms with proofs in the logic [GLT90]. For instance, the rule Modus-Ponens

$$\frac{A \Rightarrow B \quad A}{B}$$

corresponds to the typing rule of the application shown in Eq. (B.3). In a similar way as lambda calculus can be extended with constants and constructs, type systems can be very sophisticated to capture many properties of the underlying language [Pie02], drawing deep connections with expressive logics. In the context of a quantum extension to the lambda calculus, such a relevant logic turns out to be linear logic [Gir87]: this is discussed in Section B.3.3.

B.3.2 Quantum Extension to the Lambda Calculus

The idea behind the quantum lambda calculus is to offer an interface to the quantum coprocessor. To this end, it is natural to endow the language with two constant types `bit` and `qbit`, respectively standing for classical Boolean values and quantum bits. Three term constants can then be added: `meas` for measuring a qubit, `qinit` for initializing new qubits, and a family of constants U, ranging over a set of unitary gates.

The question is how to incorporate qubit objects in the language. A naïve approach consists in adding one constant for each quantum state: one could then write for instance

$$\lambda f.\lambda g.(f\,|0\rangle)(g\,|1\rangle) \tag{B.4}$$

The problem with such an approach is entanglement: What if the two-qubit system in state $|01\rangle$ used in Eq (B.4) where instead in state $\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle)$? As proposed by van Tonder [Ton04], one could imagine a quantum superposition of terms. But this turns out to be in fact equivalent to simply storing the quantum state on the side, and using pointers to qubits in the term, as follows:

$$\left[\ \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \quad |xy\rangle\,, \quad \lambda f.\lambda g.(f\,x)(g\,y)\ \right] \tag{B.5}$$

with $|xy\rangle$ being a compact representation for a function sending x to qubit position 0 and y to qubit position 1.

In a series of papers [SV05, SV06, Val04, SV09, Val08], we follow this now standard procedure to define a quantum lambda calculus and its operational semantics. A program is then a triple $[Q, L, M]$ mimicking a simple quantum coprocessor where gates are sent one-at-a-time. In the triple, the element Q is the state of the quantum memory, M is a lambda-term with free variables standing for pointers to qubits in the memory, and L is a *linking function* addressing each pointer to their qubit position in the memory. Due to the nature of the measurement, the evaluation ends up being probabilistic: there is the need for a choice of *reduction strategy*, since tossing a coin and duplicating the result is not the same thing as duplicating the coin and tossing (once) each copy. In the case of the quantum lambda calculus, following effectful

higher-order languages such as Ocaml, the original choice has been a call-by-value reduction strategy: an argument is reduced to a value before being passed along to the function.

This standard abstract machine and reduction procedure is described in more details in Section D.1.2 together with our later contributions.

B.3.3 Linear Type System

In the quantum lambda calculus, qubits have a special property: they are *non-duplicable*. Indeed, if the function

$$\lambda x.(CNOT\, x)\, x$$

inputting a qubit and passing it to the control-NOT both as control and as active qubit is not valid. Similarly, the behavior of the term

$$\lambda x.(M(\mathtt{meas}\, x))(U\, x))$$

heavily depends on the evaluation ordering of arguments. In order to feature the usual safety properties, a type system for the quantum lambda calculus has to enforce non-duplicability, i.e. linearity of qubits.

Because of the higher-order nature of the language, non-duplicability is not restrained to qubits. For instance, the term

$$(\lambda x.(\lambda f.f\, x))(\mathtt{qinit}\, \mathtt{tt})$$

(where $\mathtt{tt}$ stands for the Boolean True) initializes a new qubit and makes a function using this qubit. The function is thus non-duplicable as it contains a qubit inside its body. Non-trivial examples can be constructed based on the teleportation algorithm [SV06] or the Bell experiment [Val11].

A suitable resource-sensitive logic is *linear logic* [Gir87]: objects are strictly linear by default, and a special type constructor "!" is added to the logic to tag duplicable and erasable elements. The original type systems for the quantum lambda calculus builds upon linear logic: In Section D.1.2, we present an instantiation following intuitionistic affine linear logic.

B.3.4 Towards a Denotational Semantics

A *denotational semantics* is a mathematical—or categorical—models characterizing the behavior of programs [Sto77, Sch86, LS89]. A denotational semantics attaches to each type a mathematical space—or an object of a category—and to each well-typed term a suitable function —or morphism.

The strictly linear fragment. In the context of quantum computation, the natural mathematical framework consists in density matrices and superoperators —or more generally, positive matrices and completely positive maps—. Capitalizing on the Choi theorem [Cho75], Selinger [Sel04a, Sel04b] describes a (concrete) compact closed category based on cones of positive matrices and completely positive maps. This category has been shown to provide a fully-abstract model for strictly linear quantum computation in [SV08a]. However, as it is based on finite-dimensional vector spaces it cannot handle inductive types such as natural numbers or lists. Similarly, it is not expressive enough to model the type constructor "!": this will be the subject of Section D.2.

Towards duplication. In [SV08b], we made a preliminary, abstract proposal for the structure required for a model of a full quantum lambda calculus. The proposed structure is based on 2 categories and a strong monad:

- A symmetric monoidal category C, standing for the computations available inside the quantum coprocessor,

- A cartesian closed category $\mathcal{D}$ for classical, effect-free higher-order computation,
- A strong monad on $\mathcal{D}$ abstracting the probabilistic side-effect

The two categories $\mathcal{C}$ and $\mathcal{D}$ form a linear-non-linear model [Bie93, Ben94b], therefore giving rise to a semantics for the "!" operator as a comonad. Linear-non-linear models form the root of all existing semantics for state-of-the-art circuit-description languages [PRZ17, RS18b, LMZ18, FKS20]. Accommodating duplication and circuit construction is the topic of Chapter D.

Chapter C

Quantum Languages and Compilation Toolchain

At the turn of the 2010s, quantum coprocessors started to be considered mature enough for quantum algorithms to be competitive compared to purely classical ones [QCS].

The problem was to connect two distinct lines of work. On one side, the design of quantum algorithms, focusing on their asymptotic behavior, and the other quantum programming languages, very minimalist at the time. Furthermore, leaning toward the concrete use of quantum algorithms requires to conceptualize the compilation of the language onto concrete hardware. A quantum compilation toolchain needs to take into account the constraints of the coprocessor: the small memory size, the structure of the memory, and possibly the noise of the backend. Because of the many ways quantum algorithms are described, a compilation frameworks has to be equipped with robust methods for synthesizing and optimizing circuits out of classical specifications—whether provided as matrices or given as classical code. Finally, the counter-intuitive behavior of quantum computation added to the difficulty of testing programs hints toward the development of a dedicated set of formal methods and analysis techniques for quantum program.

This chapter is devoted to a presentation of the author's work on these aspects: design of a scalable quantum programming language, circuit synthesis techniques, and analysis tools from a practical point of view. Each of them covers a section.

- Section C.1 presents our main contribution on the topic of scalable quantum programming language: the design of the language Quipper [GLRSV13b]. We first discuss the concept of circuit-description language and how it offers a sound, formal paradigm for interacting with the coprocessor. We then introduce Quipper, a domain-specific language embedded in Haskell and following this principle. We finally present a use-case enlightening the effectiveness of the approach: The logical resource estimation of an instance of the Quantum Linear System Algorithm [SVMABC17].

- Section C.2 discusses three of our lines of works concerned with circuit synthesis and optimization. We first present a technique, novel at the time, to automatically construct an oracle (the circuit U_f of Figure B.3) from the code of a classical function (the function f of Figure B.3) [Val16]. We then discuss circuit synthesis out of the description of a unitary matrix—an array of complex numbers [BBVA20]. We finally turn to the question of the use of the ZX calculus as a tool for describing and optimizing quantum circuits [BPV21].

- Section C.3 considers the problem of quantum program certification. Testing being hard —if not impossible— when manipulating quantum information, certifying that a quan-

tum program behaves as expected requires formal methods and proof techniques. In this section, we discuss the problems this raises and argue that deductive verification is a suitable technique for the problem. We then present our contributions: a Floyd-Hoare logic for recursive quantum programs [XVY21], and Qbrick, our proposal for proving properties of quantum programs in a scalable manner [CBBPV21].

My contributions to the field are summarized by the sectioning of the chapter. Section C.1 covers the series of works on the design of quantum programming languages derived from my post-doc in the US in 2011-2013. Section C.2 highlights some of the results I participated in developing, in particular with two of my former Ph.D students: Timothée Goubault de Brugière, Ph.D student in CIFRE co-supervised with Marc Baboulin (LMF) and Cyril Allouche (Atos), and Agustin Borgna, Ph.D student co-supervised with Simon Perdrix (LORIA). Finally, Section C.3 skims through the problem of specification and verification, and my contributions to the field, some of it coming from a collaboration with the quantum group at CEA-LIST/LSL. The collaboration is still ongoing with a Ph.D student: Jérome Ricciardi.

C.1 Quipper: a Circuit-Description Language

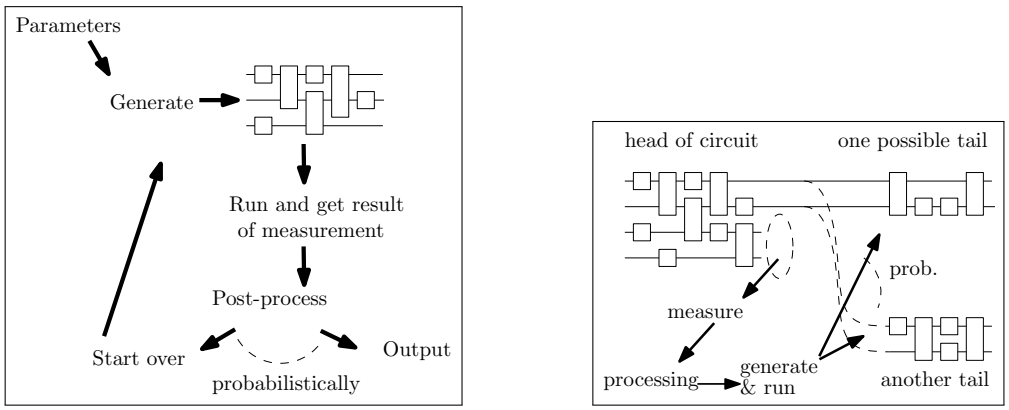
This section is devoted to one of our main contributions: the design of Quipper, the first scalable quantum programming language. Before Quipper, the state of the field, described in Section B.2, showed little connection between algorithm use-cases and quantum programming languages. This was a serious roadblock for investigating the concrete applicability of quantum algorithms.

The main formal realization we made while working on Quipper is the fact that realistic quantum algorithms require a *circuit-description* language with both low-level and high-level circuit constructors. With Quipper, we propose a formal, sound setting for representing quantum programming, opening the door to program verification and certification. This section is devoted to the presentation of Quipper. Section C.1.1 discusses the main design principles we developed. Section C.1.2 presents Quipper, and Section C.1.3 sketches one of our contribution using Quipper: the logical resource estimation of an instance of a quantum algorithm for solving linear systems of equations.

From 2011 to 2013 I was postdoc at the University of Philadelphia, in the US, employed by the large pan-American QCS project [QCS] funded by IARPA. The project spanned physics and computer science; I was hired to work on the language aspect.
One of the goals of the QCS project was to provide a *concrete* logical resource estimation for quantum algorithms. Seven algorithms were chosen by IARPA:

1. [CCDFGS03] to find a labeled node in a graph;
2. [ACRŠZ10] to evaluate a NAND formula;
3. [Hal07] to approximate the class group of a real quadratic number field;
4. [WBA11] to compute the ground state energy level of a particular molecule;
5. [HHL09, Amb12, CJS13] to solve a linear system of equations;
6. [Reg04] to choose the shortest vector among a given set;
7. [MSS07] to exhibit a triangle inside a dense graph.

The objective was to span a reasonably representative set of the existing algorithms of the time[a]. The chosen algorithms make use of a wide variety of quantum primitives such as amplitude amplification, quantum walks, quantum Fourier transform (QFT), quantum phase estimation (QPE), quantum simulation, *etc*. Several of the algorithms also require the implementation of sophisticated classical oracles. The starting point



(a) Static scheme

(b) Dynamic scheme

Figure C.1: Workflows for quantum algorithm

for each of our algorithm implementations was a detailed description of the algorithm provided by IARPA.
As part of the project, we developed Quipper as a tool to answer the particular problematics of *coding* the aforementioned algorithms in the context of the IARPA project, and the research spurred a series of papers: [GLRSV12, GLRSV13b, GLRSV13a, SRSV14, VRSAS15, SVMABC17]. Along the line we conceptualized the language design principles presented in Section C.1.1, and we used Quipper for concrete logical resource estimation. My contribution on the latter part is presented in Section C.1.3.

[a] Note however how the later trend of variational algorithms is —obviously— not represented.

C.1.1 Discussion: Quantum Programming Language Design

In Section B.1.4, circuits were merely seen as sequences of elementary gates. However, in most quantum algorithms circuits are more complex structures, built compositionally from smaller sub-circuits and circuit combinators. If they are usually static objects, buffered until complete before being flushed to the quantum coprocessor, in some algorithms, circuits are even *dynamically* generated: the tail of the circuit depending on the result of former measurements.

In this section, we discuss the high-level structure of quantum algorithms, the requirements for a quantum programming language, and review some of the existing proposals.[1]

C.1.1.1 Structure of Quantum Algorithms

The usual model for quantum computation was discussed in Section B.1.4: a classical computer controls a quantum coprocessor, whose role is to hold a quantum memory. A programmatic interface for interacting with the coprocessor is provided to the programmer sitting in front of the classical computer. The interface gives methods to send instructions to the quantum memory to allocate and initialize new quantum registers, apply unitary gates on qubits, and eventually perform measurements. If the set of instructions is commonly represented as a circuit, it is merely the result of a *trace of a classical execution* of a classical program on the classical computer.

Figure C.1 presents two standard workflows with a quantum coprocessor. In Figure C.1a, the classical execution inputs some (classical) parameters, performs some pre-processing, gen-

[1] This section is heavily inspired from my own contribution in [CBBPV21].

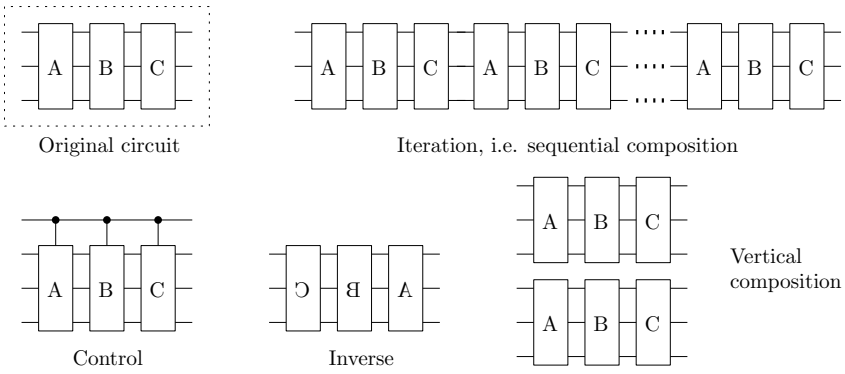


Figure C.2: Standard Circuits Combinators

erates a circuit, sends the circuit to the coprocessor, collects the result of the measurement, and finally performs some post-processing to decide whether an output can be produced or if one needs to start over. Shor's factoring algorithm [Sho97] and Grover's algorithm [Gro96] fall into this scheme: the circuit is used as a fancy probabilistic oracle. Most of the recent variational algorithms [MRBA16, CABB+21] such as VQE [PMSY+14] or QAOA [FGGLLP01, FGG14] also fall into this scheme, with the subtlety that the circuit might be updated at each step. The other, less standard workflow is presented in Figure C.1b. In this scheme, the circuit is built "on the fly", and measurements might be performed on a sub-part of the memory along the course of execution of the circuit. The latter part of the circuit might then depend on the result of classical processing in the middle of the computation. One can for example cite the Unique Shortest Vector algorithm [Reg04], or the more standard repeat-until-success procedures [LBK05, PS14].

Understanding quantum circuits as a by-product of the execution of classical programs shines a fresh light on quantum algorithms. Unlike a naive interpretation, a quantum algorithm cannot be identified with a quantum circuit. Instead, in general, at the very least a quantum algorithm describes a *family* of quantum circuits. Indeed, consider the setting of Figure C.1a. The algorithm is fed with some parameters and then builds a circuit: the circuit will depend on the shape of the parameters. If for instance we were using Shor's factoring algorithm, we would not build the same circuit for factoring 15 or 110,423,192,017. A quantum programming language should therefore be able to describe *parametric families* of circuits.

The circuits described by quantum algorithms are potentially very large. We show for instance in [SVMABC17] how a concrete instance of the HHL algorithm [HHL09] for solving linear systems of equations can require as many as $\sim 10^{40}$ elementary gates, if not optimized—see Section C.1.3 for details. To handle the scalability, quantum algorithms describe circuits by composing sub-circuits—possibly described as list of elementary gates but not only—using high-level circuit combinators. These combinators build circuits by (classically) processing possibly large sub-circuits. Some standard combinators are shown in Figure C.2 (where we represent inverse with reflected letters). Note that there is a distinction to be made between a combinator, applied on a sub-circuit, and its semantics, which is an action on each elementary gate. Combinators are abstractions that can be composed to build larger combinators, such as the one presented in Figure C.3 built from inversion, controlling and sequential composition.

C.1.1.2 Requirements for Quantum Programming Languages

Any scalable quantum programming language should therefore allow the following operations within a common framework.

- Manipulation of quantum registers and quantum circuits as first-class objects. The pro-

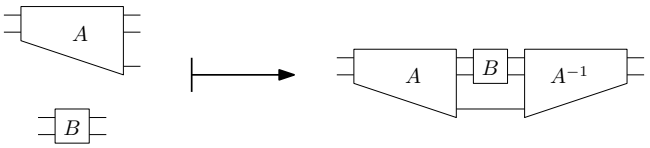


Figure C.3: Example of derived circuit combinator

grammer should both be able to refer to "wires" in a natural manner and handle circuits as independent objects.

- Description of *parametric families* of quantum circuits, both in a procedural manner as sequence of operations—gates or subcircuits—and in an applicative manner, using circuit combinators;
- Classical processing. In our experience [GLRSV13b], quantum algorithms mostly consists of classical processing: processing the parameters, building the circuits, processing the result of the measurement.

This broad description might call for refinements. For instance, some of the classical processing might be performed on the quantum coprocessor —such as the simple classical controls required for quantum error correction. The level of classical processing performed on the classical computer —therefore requiring communication through the interface— and performed on the quantum coprocessor —requiring a more or less sophisticated device— is dependent on the physical implementation. If some recent proposals such as Quingo [Qui20] discuss the design of quantum programming languages aware of the two levels of classical processing —in and out of the coprocessor— this is still work in progress.

C.1.1.3 Review of the Existing Approaches

Most of the current existing quantum programming languages follow the requirements discussed in Section C.1.1.2. In this section, we review some typical approaches followed both in academic and in industrial settings. This review is by no means meant to be exhaustive: its only purpose is to discuss the possible strategies for the design of quantum programming languages (QPLs).

When designing a realistic programming language from scratch, the main problem is the access to existing libraries and tools. In the context of quantum computation, one would need for instance to access the filesystem, make use of specific libraries, *etc.* In order to quickly come up with a scalable language, the easiest strategy consists in *embedding* the target language in a host language. Indeed, a quantum programming language can be seen a domain-specific language (DSL), and it can be built over a regular language. One can then rely on the possibly well-maintained and optimized compiler or interpreter of the host language.

If the advantages of working inside a host language are clear, there are two main drawbacks, The first one is the potential rigidity of the host language: there might be constructs natural to the DSL that are hardly realizable inside the host language. The second drawback has to do with the compilation toolchain: the shallow embedding of the DSL makes it impossible to access its abstract syntax tree (AST), therefore rendering its manipulation impossible.

Embedded QPLs. The first scalable embedded proposal is Quipper [GLRSV13b, GLRSV13a]. Embedded in Haskell, it capitalizes on *monads* to model the interaction with the quantum coprocessor. Quipper's monadic semantics is meant to be easily abstracted and reasoned over: it is the subject of Section C.1.2. Since Quipper, there has been a steady stream of embedded quantum programming languages, often dedicated to a specific quantum coprocessor or attached to a specific vendor, and mostly in Python: Qiskit [Qis] and ProjectQ [SHT18]

for IBM, CirQ for Google, Strawberry Fields [KIQBAW19] for Xanadu, PyQuil and Forest for Rigetti [SCZ17], AQASM for Atos [Ato22], *etc.* From a language-design perspective, most of these approaches make heavy use of Python objects to represent circuits and operations. The focus is on usability and versatility more than safety and well-foundness.

Standalone QPLs. On the other side of the spectrum, quantum programming languages have been designed as standalone languages, with their own parser, and therefore abstract syntactic tree. Maybe the first proposed scalable language was Ömer's QCL [Öme03]. Ömer experimented with several features such circuit-as-function, automatic inversion and oracle generation. However, due to its non-modular approach the language did not have successors. Liqui|⟩ [WS14] and its sequel Q# [SGTA+18], developed by Microsoft are good examples of an attempt at building a standalone language while keeping a tight link with an existing programming environment: Q# is based on the F# framework, making it possible to easily "use" library functions from within a Q# piece of code. On the other hand, Q# has his own syntax and type system. This makes it possible to capture run-time errors specific to quantum computation. ScaffCC [JPKH+15] is another example of a QPL with its dedicated parser. If the language is rather low-level its compiler has been heavily optimized and experimented over, and it serves as support for a long stream of research on quantum compiler optimizations [CFM17, LFSMC20]. The last noteworthy language to cite in series is Silq [BBGV20], as it serves as a good interface with the next paragraph: aimed at capturing most of the best practice in term of soundness and safety, it is nonetheless targeted toward usability.

C.1.2 Our Proposal: Quipper

This section is devoted to the presentation of the language Quipper: a circuit-description language based on the design principles described in Section C.1.1. Quipper is a language embedded in the host language Haskell and uses a monadic semantics to enforce the desired operational semantics—that is, circuit construction. Section C.1.2.1 quickly presents what is a monadic semantics and Section C.1.2.2 describes how Quipper makes use of it.

C.1.2.1 Circuit Construction with a Monadic Approach

The solution devised by Quipper consists in relying on a special language feature from Haskell called *monad*. A monad is a type operator encapsulating a side effect. Consider for instance a probabilistic side effect. The monad is regarded as a type operator, e.g. `P`. There are two classes of terms: terms without side-effect, with types e.g. `Bool`, or `Int`, and terms with side-effect, with types e.g. `P(Bool)`, standing for "term evaluating to a boolean, possibly with a probabilistic effect", or `P(Int)`. The operator `P(-)` captures the probabilistic side effect.

A monad comes with two standard maps: `return` regards a value as a "term with a (trivial) side-effect", and `eval`, for applying a function to an effectful term. For `P` we would have

$$\texttt{return} :: \texttt{a} \longrightarrow \texttt{P(a)} \qquad \texttt{eval} :: (\texttt{a} \longrightarrow \texttt{P(b)}) \longrightarrow \texttt{P(a)} \longrightarrow \texttt{P(b)}$$

A few equations have to be satisfied by `return` and `eval` for them to describe a monad. For instance, `eval return` is the identity on `P(a)`. There can of course be more operations: for instance, we can add to the signature of `P` an operator `coin` of type `() -> P(Bool)`, whose semantics would be to return `tt` or `ff` with equal probability.[2].

A nice property of monads is that effectful operations can be written with syntactic sugar in an imperative style, as follows.

[2] In Haskell, the unit type is denoted with `()`.

```
do
  x <- coin ()
  if x then return 0 else return 1
```

The program above is of type `Int` and is equal to

$$\texttt{eval}\ (\lambda \texttt{x.if x then return 0 else return 1})\ (\texttt{coin}\ ())$$

once the syntactic sugar has been removed.

Following this approach, quantum computation can be understood as a side-effect: it combines both (1) read/write effects, since gates are sent to the coprocessor, and results of measurements are received; (2) Probabilistic effects, since measurement is a probabilistic operation. The first attempt at formalizing this monad is Green's quantum IO monad [AG09]: it has then been further developed in Quipper [GLRSV13b], subject of Section C.1.2.2.

C.1.2.2 Design principles for Quipper

Quipper is built as an embedded language in Haskell. It makes heavy use of Haskell's type classes [WB89] and type families [KJS10] to enhance parametricity. Monads freely come in Haskell as a particular type class. As discussed in Section C.1.2.1, Quipper's operational semantics relies on a specific *monad* encapsulating circuit construction: the `Circ` monad. The interaction with the coprocessor can be modeled with an I/O interface: gates are emitted, while branching occurs following a read operation. The `Circ` monad is then based on an inductive construction akin to the following.

```
data CircIO a =
  Empty a
| Unit Gate
| Meas (Bool -> CircIO a)
```

Circuits in Quipper feature wires of type `Qubit` and `Bit` —i.e. bit-wires resulting from a measure. The signature of the `Circ` monad includes the following operations.

```
qinit :: Bool -> Circ(Qubit)
measure :: Qubit -> Circ(Bit)
dynamic_lift :: Bit -> Circ(Bool)
had :: Qubit -> Circ(Qubit)
```

The coin-toss of Section C.1.2.1 can be written as

```
coin () = do
  q <- qinit True
  q' <- had q
  r <- measure q'
  dynamic_lift r
```

The function `coin` is of type `() -> Circ(Bool)`: running `coin` will merely generate a computation —a circuit to be executed— waiting to be executed.

Thanks to the monadic encapsulation, circuits can be manipulated within Haskell. For instance, inversion and control can be coded in Haskell as circuit combinators with the following types.

```
inverse :: (a -> Circ b) -> (b -> Circ a)
control :: (a -> Circ b) -> ((a,Qubit) -> Circ (b,Qubit))
```

One can also define operators to interact with circuits, such as

```
count :: (Circ a) -> Int
simulate :: (Circ [Qubit]) -> Prob [Bool]
```

where `count` returns the number of gates in the circuit, and `simulate` classically emulates the input circuit followed with a measurement and returns the probability distribution.

The strength of Haskell's monadic approach is the ability to capture *parametric families* of circuits within the framework. For instance, a tower $H^{\otimes n}$ of Hadamard gates (parameterized by n) can be defined as

```
  mapM hadamard :: [Qubit] -> Circ [Qubit]
```

where `[Qubit]` stands for the type of list of qubits. When fed with one specific list of qubits, the program generates the corresponding circuit. This program is then indeed the description of a *family* of circuits.

C.1.3 Use-Case: Logical Resource Estimation of the QLS Algorithm

In this section, we present one of the concrete application of the language Quipper: the first complete *logical resource estimation* for one particular, concrete algorithm.

Indeed, before 2013, quantum algorithms were still theoretical apparatuses meant to study the inherent asymptotic complexity of problems. While moving towards concrete use-cases, one of the problem that arises is the discrepancy between the theoretical efficiency of an algorithm and its particular implementation on a concrete problem instance. In particular, as a quantum algorithm builds a circuit, what is the size of this circuit, provided a given problem instance?

In a journal publication [SVMABC17], we perform a logical resource estimation for the *quantum linear system* problem (QLS), for solving linear systems of equations. If the original algorithm has been laid out by Harrow, Hassidim, and Lloyd [HHL09]—thus the common name for the algorithm: HHL—the algorithm went through several refinements: first by Ambainis [Amb12] and then by Clader *et al.* [CJS13]. The latter was the focus of the work that was analyzed in the journal publication [SVMABC17].

Statement of the Problem The QLS algorithm aims at solving a system of linear equations of the form $A\vec{x} = \vec{b}$, where A is a Hermitian $N \times N$ matrix of complex numbers, $\vec{b}$ is a $\mathbb{C}$-vector of dimension N, and $\vec{x}$ is the unknown vector. Solving the equation morally corresponds to inverting A.

The basic idea of the QLS algorithm is the following. Provided that λ_i and u_i are respectively the eigenvalues and eigenvectors of A, if $\vec{b} = \sum_i \beta_i u_i$, with suitable side conditions the solution of the equation is simply

$$\vec{x} = \sum_{i=1}^{N} \frac{\beta_i}{\lambda_i} u_i.$$

The algorithm then relies on several non-trivial pieces: the Quantum Phase Estimation (QPE) to retrieve the λ_i's; oracles for A, $\vec{b}$ and the inversion; an Hamiltonian simulation [BACS07] to build a circuit for e^{itA}.

Summary of the Complexity Analysis. The original QLS algorithm and its subsequent refinements [HHL09, Amb12, CJS13] leave aside the implementation details and only focus on the general asymptotic complexity of the algorithm. It uses several parameters of the problem instance: the size N of the matrix ; the maximal error allowed ε ; the *sparseness* d of the matrix A, that is, the maximum number of non-zero entries per row and column ; the *condition number* κ, defined as the ratio between the largest and smallest eigenvalues of A. The smaller κ is, the closer it is to be invertible: κ gives information on the stability of the solution $\vec{x}$.

That being said, the best known classical algorithm for solving linear systems of equations are based on the conjugate gradient method [She94, Saa03], and they have a run-time complexity of $O(Nd\kappa \log(1/\varepsilon))$. By contrast, the HHL algorithm [HHL09] attains

$$\widetilde{O}(\kappa^2 d^2 \log(N)/\varepsilon) \tag{C.1}$$

(where $\widetilde{O}(-)$ suppresses more slowly growing terms compared to $O(-)$). Provided that the matrix is well-conditioned and sparse enough, we theoretically get an exponential improvement over the classical algorithm. The improvement proposed by Clader *et al.* [CJS13] provides a complexity

$$\widetilde{O}(\kappa d^7 \log(N)/\varepsilon^2). \tag{C.2}$$

For very sparse matrices, this algorithm is likely to beat the original HHL algorithm.

However, all of these "big-O" complexities are blind to the structure of the concrete description of the matrix A and of the vector $\vec{b}$: they do not help with logical resource estimates for concrete problem instances.

Logical Resource Estimation for a Problem Instance. In the project QCS—and the resulting paper [SVMABC17]—we applied the QLS algorithm to a linear system of equations coming from the discretization of Maxwell's equations using the finite-element method (FEM), to determine the electromagnetic scattering cross-section of a specified target object [CJV93]. FEM tends to generate *sparse* matrices, one of the conditions for the QLS algorithm.

To decide on the size N of the matrix, using the big-O estimates, we came to $N \sim 4 \cdot 10^7$ as the "cross-over point" at which the quantum algorithm would beat the classical algorithm. We chose the somewhat larger value $N = 332,020,680$ to stay on the safe side: it is reasonable to expect such a problem size to be hardly tractable classically. The oracles for A and $\vec{b}$ can be derived from the problem instance. Their description is *classical*: see e.g. Fig. C.4 for a piece of the specification of the oracle $\vec{r}$, coded in Haskell. Note in particular how this requires high-level libraries such as trigonometric functions.

Discussion For the chosen problem instance, the other parameters governing the complexity yield $d = 7$, $\kappa = 10^4$ and $\varepsilon = 0.01$. The logical resource estimation for $N = 332,020,680$ is shown in [SVMABC17, Table 2, p. 42]: the circuit generated by the algorithm consists of $2.37 \cdot 10^{29}$ elementary gates amongst H, T, S, X, Z and *CNOT*, and $3 \cdot 10^8$ total qubits (most of them being ancillas required for the oracles). Not counting the oracles, the number of gates falls to $3.34 \cdot 10^{25}$ with only 281 qubits. The bottom line is that a big-O resource estimate is not enough for deciding on the usability of a particular quantum algorithm. Another conclusion is that optimization techniques are going to be an essential tool in a quantum compilation toolchain.

The analysis performed in [SVMABC17] was novel at the time: it was the first concrete analysis of the resources needed to run a quantum algorithm, without relying on "big-O" estimates. Since the coding of the QLS algorithm in Quipper there has been a steady stream of research on Hamiltonian simulation, see e.g. [BCCKS14, BCK15, Low19].

```
calcRweights y nx ny lx ly k theta phi =
    let (xc',yc') = edgetoxy y nx ny in
    let xc = (xc'-1.0)*lx - ((fromIntegral nx)-1.0)*lx/2.0 in
    let yc = (yc'-1.0)*ly - ((fromIntegral ny)-1.0)*ly/2.0 in
    let (xg,yg) = itoxy y nx ny in
    if (xg == nx) then
        let i = (mkPolar ly (k*xc*(cos phi)))*
                (mkPolar 1.0 (k*yc*(sin phi)))*
                ((sinc (k*ly*(sin phi)/2.0)) :+ 0.0) in
        let r = ( cos(phi) :+ k*lx )*((cos (theta - phi))/lx :+ 0.0) in i * r
    else if (xg==2*nx-1) then
        let i = (mkPolar ly (k*xc*cos(phi)))*
                (mkPolar 1.0 (k*yc*sin(phi)))*
                ((sinc (k*ly*sin(phi)/2.0)) :+ 0.0) in
        let r = ( cos(phi) :+ (- k*lx))*((cos (theta - phi))/lx :+ 0.0) in i * r
    else if ( (yg==1) && (xg<nx) ) then
        let i = (mkPolar lx (k*yc*sin(phi)))*
                (mkPolar 1.0 (k*xc*cos(phi)))*
                ((sinc (k*lx*(cos phi)/2.0)) :+ 0.0) in
        let r = ( (- sin phi) :+ k*ly )*((cos(theta - phi))/ly :+ 0.0) in i * r
    else if ( (yg==ny) && (xg<nx) ) then
        let i = (mkPolar lx (k*yc*sin(phi)))*
                (mkPolar 1.0 (k*xc*cos(phi)))*
                ((sinc (k*lx*(cos phi)/2.0)) :+ 0.0) in
        let r = ( (- sin phi) :+ (- k*ly) )*((cos(theta - phi)/ly) :+ 0.0) in i * r
    else 0.0 :+ 0.0
```

Figure C.4: Part of the specification of the $\vec{r}$ oracle

C.2 Circuit Synthesis and Optimization

A quantum circuit serves two purposes. First of all, it serves as the description of a linear operation on the memory state space. Furthermore, it gives a procedure to implement this linear map, with informations on the resources required to realize it.

Along the description of a quantum circuit, some subcircuits might only be specified by the linear map they implement. The designers of the quantum algorithm relies on an external authority to attest that the corresponding subcircuit is indeed realizable within the required framework, and leaves the generation of the quantum circuit to a hypothetical compiler.

Circuit synthesis tools are therefore crucial tools for a quantum compilation toolchain. This section explores three cases. Section C.2.1 considers the problem of synthesizing oracles. A typical oracle is given as a classical description such as the structure of a graph to explore or an arithmetic operation to perform. The section presents the solution we implemented for Quipper, automatically turning a classical code into a reversible circuit. Section C.2.2 focuses on the synthesis of circuits corresponding to linear maps given as matrices of complex numbers. We present two solutions based on numerical techniques to answer the problem, and discuss the sizes of the generated circuits. Finally, Section C.2.3 sketches our contribution for circuit generation out of a ZX description, in an hybrid quantum and classical setting.

> Section C.2.1 is devoted to my contribution on automatic generation in Quipper for the QLS algorithm, to be able to handle the oracle of Figure C.4. Section C.2.2 presents works stemming out of the Ph.D supervision of Timothée Goubault de Brugière, CIFRE co-supervised by Marc Baboulin (LRI) and Cyril Allouche (Atos). Section C.2.3 discusses results from the Ph.D of Agustin Borgna, co-supervised with Simon Perdrix (LORIA, Nancy).

C.2.1 Circuit Synthesis from Oracle Specification

My first contribution to the field of circuit synthesis consists in the development of an automated procedure to translate a functional program working with Boolean values to a circuit realizing the same computation. Written in Haskell using Template Haskell [TempHask], the tool inputs a Haskell, first-order function and produces an object in the `Circ` monad, encapsulating a circuit realizing the input function. The tool is one of the libraries available with Quipper, and it has been extensively used for the oracle of the QLS algorithm, as discussed in Section C.1.3. In particular, it was what made it possible to realize the trigonometric functions using fixed-point real numbers (see e.g. Figure C.4).

The tool makes heavy use of Haskell's monad feature. I developed the formalism in a publication [Val16]: This section is devoted to its presentation.

Irreversible to Reversible Computation The study of reversible computation and how it relates to irreversible computation has been a subject of research since the 1960s. Landauer [Lan61] follows a trend of research discussing how irreversible computation dissipates energy (and heat) and how reversible computation could be a way to reduce computational energy consumption. In the following years, several models of reversible computation have been proposed: reversible Turing machines [Ben73], reversible cellular automata [Moo62, Tof77, Dur02], reversible boolean circuits [Tof80b], billiard ball models [FT82], *etc*. Various concrete, physical reversible processors have also been proposed in the literature, aiming at being more efficient than their irreversible counterparts [Hal92, Fra99]. The interest for the subject has not declined [Ben00, Ada02, FBCH+20], as for instance shown by the recent ICT COST Action IC1405 [IC1405] and the series of conferences on reversible computation [RC21].

Although the two subjects stem from distinct origins, reversible computation has seen an unexpected use in quantum computation. Indeed, as discussed in Section C.1.1 one of the necessary building block of quantum algorithms is *oracles*: unitary maps realizing classical, irreversible computations. As a unitary map is first and foremost a reversible operation, all of the machinery developed for reversible computation can be used for oracle synthesis. And indeed, in the literature most of the complexity analysis of quantum algorithm relies on the seminal papers of Fredkin, Toffoli and Bennett [Tof80b, FT82, Ben73] to assert the existence of efficient oracle synthesis.

Fredkin and Toffoli [Tof80b, FT82] were amongst the first ones to state the problem of reversible computation using a circuit formalism. Within this framework, they integrate the so-called *Landauer's embedding* with *Bennett's trick* to turn a classical, irreversible function

$$f : \mathtt{bit}^n \to \mathtt{bit}^m$$

into a reversible circuit in the shape of the oracle shown in Figure B.3, computing

$$\begin{array}{rccc} \widetilde{f}: & \mathtt{bit}^n \times \mathtt{bit}^r \times \mathtt{bit}^m & \to & \mathtt{bit}^n \times \mathtt{bit}^r \times \mathtt{bit}^m \\ & (\vec{x},\ \vec{0},\ \vec{y}) & \mapsto & (\vec{x},\ \vec{0},\ \vec{y} \oplus f(\vec{x})). \end{array} \tag{C.3}$$

Recall that $\oplus$ stands for bitwise XOR boolean gate. Provided that the function f is described by a (boolean) formula, Toffoli shows that the function $\widetilde{f}$ can be realized by a circuit of linear size compared to the number of logical gates in the description of f. The register of r bits in the middle is used for storing intermediate results, and the computation sets it back to $\vec{0}$ when done.

The idea consists in using Bennett's trick [Ben73] to first build a (reversible) circuit $\widehat{f}$ computing the *Landauer embedding* of f

$$\begin{array}{rccc} \widehat{f}: & \mathtt{bit}^n \times \mathtt{bit}^r \times \mathtt{bit}^m & \to & \mathtt{bit}^n \times \mathtt{bit}^r \times \mathtt{bit}^m \\ & (\vec{x},\ \vec{0},\ \vec{0}) & \mapsto & (\vec{x},\ \text{garbage},\ f(\vec{x})) \end{array}$$

in a compositional manner. Note that compared to Eq. (C.3), the middle register is not cleaned after use, rendering the computation irreversible if we were to throw away the garbage. However, as discussed below the map $\widehat{f}$ is only built from reversible components: one can recover the map $\widetilde{f}$ of Eq (C.3) using the construction shown in Figure C.5.

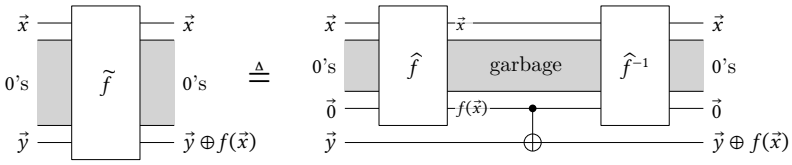


Figure C.5: Bennett's trick

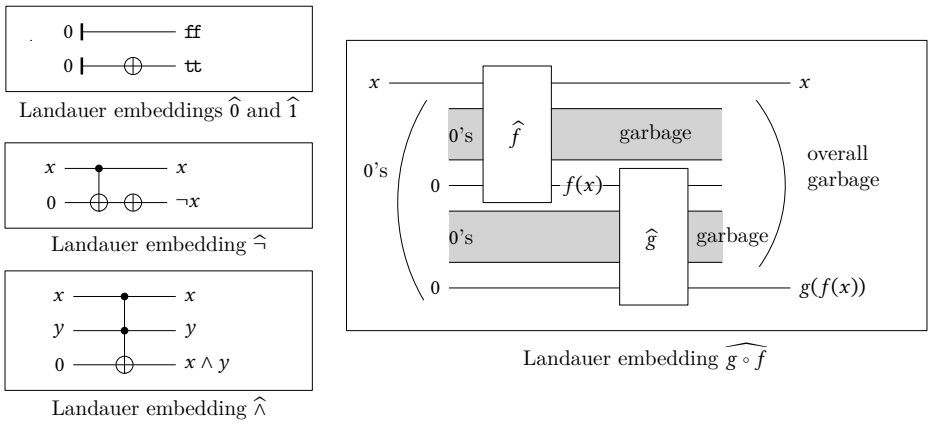
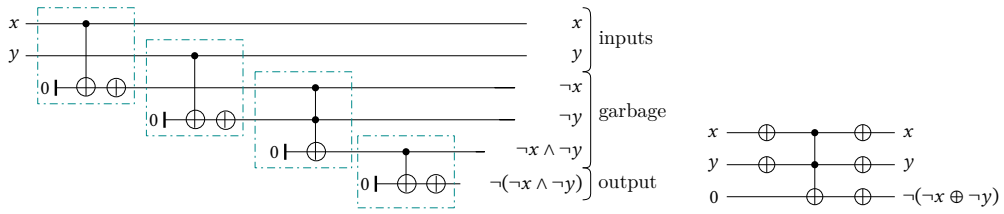


Figure C.6: Landauer embeddings of elementary logical blocks

To understand how to compositionally build $\widehat{f}$, assume that $m = 1$, and that f is built from boolean constants, conjunction ($\wedge$), negation ($\neg$), and composition thereof. The corresponding Landauer embeddings are shown in Fig. C.6, and the embedding of the function $(x, y) \mapsto \neg((\neg x) \wedge (\neg y))$ is presented in Figure C.7a. Although it gives a verbose circuit —see the equivalent, shorter circuit in Figure C.7b— it is efficient in the sense that the size of the circuit is linear on the size of the formula: each dashed sub-circuit corresponds to one logical operator.



(a) Landauer embeddings

(b) Direct implementation

Figure C.7: Circuits for $(x, y) \mapsto \neg((\neg x) \wedge (\neg y))$

Formalization of the Specification Language. The compositional procedure presented above can be formalized and generalized to higher-order functions, with the use of a monad to store the circuit under construction. For sake of conciseness, in the following, we present

a representative subset of the language described in [Val16] —we invite the reader to consult the original paper for details.

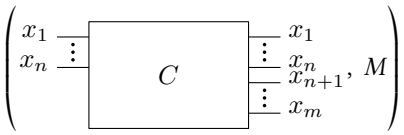
The idea consists in considering a simply-typed lambda calculus with Boolean values, and in relating two possible operational semantics for it. One semantics is the usual one, where a term of Boolean type rewrites to a Boolean value. The other one is instead a partial evaluation strategy, where the operations to perform are stored in a circuit: the circuit to be evaluated.

If the language is denoted with Λ_{bool}, we can consider for instance the definition of terms and types as

$$\begin{array}{lll} M, N & ::= & x \mid \lambda x.M \mid MN \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathsf{not} \mid \mathsf{and}, \\ A, B & ::= & \mathsf{bool} \mid A \to B. \end{array}$$

The language in [Val16] also contains lists, pairing, if-then-else and fixpoints, but these constructs do not need a substantially different approach. In any case, typing judgment are standard: they consist of a typing context , i.e. a set of typed variables $\Delta = x_1 : A_1, \dots, x_n : A_n$, and a term M of type A, written $\Delta \vdash M : A$. The typing rules for typing derivations describing valid typing judgments are as expected: the conjunction is for instance typed as $\mathsf{and} : \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool}$.

The first operational semantics is a standard call-by-value reduction strategy: we define values V, W and application contexts $S[-]$ as usual, and we for instance have the rule stating that $S[(\lambda x.M)V] \to_\beta S[M\{x := V\}]$ and that $S[(\mathsf{and}\ \mathtt{ff})\ \mathtt{tt}] \to_\beta S[\mathtt{ff}]$. The second operational semantics consists in a partial evaluation: instead of evaluating not and and, the semantics "stores" the operations to be performed inside a (reversible) circuit. The semantics is therefore based on an abstract machine of the form



(C.4)

where C is a reversible circuit consisting in wire initializations, NOT, CNOT and Toffoli gates, and where the free variables of M are within $\{x_1, \cdots, x_m\}$. For the detailed definition of circuits, we refer the reader to [Val16]. The rule for and is for instance shown in Figure C.8, where z is a fresh variable.

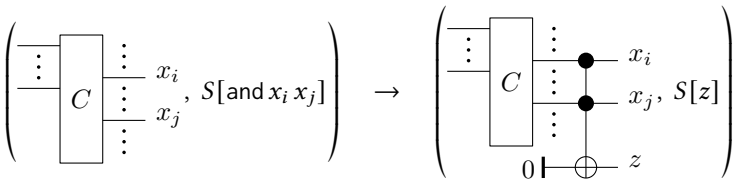


Figure C.8: Rule for and in the monadic semantics

The two semantics feature usual safety properties, and they *coincide* [Val16, Th. 17]: the circuit generated from the partial evaluation of a lambda-term M realizes the function described by M. For instance, the (non-closed) term

$$x : \mathsf{bool}, y : \mathsf{bool} \vdash (\lambda f.f\ (\mathsf{and}\ (f\ x)\ (f\ y)))\ \mathsf{not} : \mathsf{bool}$$

reduces to the circuit shown in Figure C.7a.

Automated Oracle Synthesis as Monadic Lifting. The language Λ_{bool} and the two semantics can be extended with pairs, coproducts, lists, fixpoints and tests — see [Val16]. Together with these extensions, one can internalize the definition of circuit within the language Λ_{bool}

itself. The abstract-machine semantics can then be simulated inside Λ_{bool} using a generic *monadic lifting*, close to what was proposed in [SGLH11]. It is the transposition of Haskell's monads to our language Λ_{bool} —and of the strategy used in Quipper for automatic oracle synthesis. The main characteristic of the reversible abstract-machine is to change the operational behavior of the type `bool`: the terms $\mathfrak{tt}$, $\mathfrak{ff}$ and the inline Boolean combinators do not reduce as regular lambda-terms. Instead, they trigger a side-effect, which can be simulated within a suitable monad.

The main strength of this approach—and its instantiation with Template Haskell in Quipper—is to allow the parametric description of *families of circuits*. Indeed, in Quipper, a program of type `[Bool] ->[Bool]` is lifted to `[Qubit] -> Circ [Qubit]`: the resulting code generates a circuit whose shape depends on the size of the input list. The circuit is provably *equivalent to its specification*: the classical program being lifted. Such an approach was—and to this day still is—novel.

Discussion of Other Approaches. In the literature, the design of a reversible circuit from the description of a conventional function has conventionally been approached through its truth table or properties thereof. Several methods have been designed to generate compact circuits, although only for fixed-size circuits. It would be interesting to see how to merge the two approaches.

One can for instance consider local, peep-hole optimizations based on templates [MDM03, MDM05, SM13], or rely on SAT solvers [HMSM18]. Standard classical synthesis techniques based on BDD [WD10], on LUT mapper [MSRM19] or on ESOP and Reed-Muller techniques [FTR07, GAJ06, MWD09] have been used with some success. Approaches such as QMDDs — quantum versions of binary decision diagrams— have also been considered and shown rather efficient [ZW17]. At a high-level approach, one could also make use of efficient libraries of reversible circuits for arithmetic operations [VBE96, TK05, DKRS06, TK08, WR16, RG17, Mog19] or real analysis [NTR11, WK13, NV14, SRWD17, HSRS18, HRS18].

However, if these techniques make it possible to write reversible functions with arbitrary truth tables [WGTDD08], they do not usually scale well with the size of input [HMSM18].

Synthesis of reversible circuits can be seen as a small branch of the vast area of hardware synthesis. In general, hardware synthesis can be structural (description of the structure of the circuit) or behavioral (description of algorithm to encode). In this context, Bennett's Pebble game [Ben89, LS90] have been used with success to optimize the width and depth of circuits [ARS17, BSDCM19]. Functional programming languages have been used for both structural and behavioral descriptions. On the more structural side one finds Lava [Cla01], BlueSpec [Nik04], functional netlists [PKI08], *etc.* On the behavioral side we have the Geometry of Synthesis [Ghi12], Esterel [Ber00], ForSyDe [SJA17], *etc.* Two proposals sitting in between structural and behavioral approaches are worth mentioning. First, the imperative, reversible synthesis language SyRec [WOD10], specialized for reversible circuits. Then, Thomsen's proposal [Tho12], allowing to represent a circuit in a functional manner, highlighting the behavior of the circuit out of its structure.

On the logic side, Geometry of Interaction [Gir89, Gir90, Gir95a, Gir03, Mac94, Mac95, Ghi07] is a framework that can be adapted to turn functional programs into reversible computation [Abr05, DR99], using the idea of turning a typing derivation into a reversible automaton. There have also been attempts to design reversible abstract machines and to compile regular programs into reversible computation, e.g. a reversible version of the SEMCD machine [Klu99]. More recently, the compiler REVS [PRS17] aims at compiling conventional computation into reversible circuits.

Monadic semantics for representing circuits is something relatively common, specially among the DSL community: apart from Quipper discussed in Section C.1.2, one can name Lava [Cla01], Fe-Si [BC13], *etc.* Other approaches use more sophisticated constructions, with type systems based on arrows [JS12b] in order to capture reversibility: these approaches point towards full-fledged reversible programming languages, discussed in Section E.3.1.

Method	CNOT count	Rotation count	Flops
QSD	$23/48 \times 4^n$	$9/8 \times 4^n$	19×8^n
Householder	2×4^n	2×4^n	$2/3 \times 8^n$
Lower Bound	$1/4 \times 4^n$	4^n	(unavailable)

Table C.9: Asymptotic counts for QSD and Householder decomposition

C.2.2 Circuit Synthesis from General Unitary Matrices

In the very general case, a unitary on n quantum bits is characterized by a matrix consisting of $(2^n)^2$ complex numbers. Since the matrix is unitary, the number of parameters is slightly smaller than 4^n: it is however still very much exponential on the number of qubits.

In the case of an intentional description, such as a formula or a program, this description might reduce the number of degrees of freedom of the problem and a quantum circuit of polynomial size on the number of qubits might be obtained (as e.g. in the case presented in Section C.2.1). However, for a given gate set on 1 and 2 wires, if the matrix is only given in term of its (complex) coefficients, in general the size of a quantum circuit corresponding to the matrix is bound to be exponential.

One question that can however nonetheless be posed is how to get a circuit out of this array-based description, and how to obtain it in an as efficient as possible way. In this section, we describe two results we obtained, together with Marc Baboulin, our Ph.D student Timothée Goubault de Brugière, and Cyril Allouche [BBVA19, BBVA20, Bru20].

Circuit Synthesis via Householder Transformations An operator acting on n qubits is represented by a matrix of size $2^n \times 2^n$. Generating a circuit from an arbitrary matrix is a problem that scales exponentially in n in general, and the problem of finding the smallest possible circuit for a particular operator remains challenging: Knill [Kni95] asserts the necessity of an exponential number of gates. If several decomposition techniques have been developed [BBCD+95, Cyb01, MV06, RZBB94, SBM06], in all of them the resulting number of gates however still lies within a factor of 2 of the theoretical lower bound [BM04].

In [BBVA20], the circuit synthesis problem is analyzed with a focus on both the size of the generated circuit *and* the time needed to generate it [AMMR13, HC18, MM16, NRSCM18]. We rely on a *Householder decomposition* of the matrix to construct the circuit.

In general, the Householder decomposition of any matrix A is of the form QR, where Q is a unitary and R is upper triangular. Q is obtained iteratively by zero-ing out A column by column, applying *Householder transformations* of the form

$$H_k = I_k - a_k \cdot |u_k\rangle \langle u_k|$$

for a well-chosen scalar a_k and vector $|u_k\rangle$. At the end of the procedure, Q consists in the product of the H_k's.

Thanks to the specific structure of unitary matrices, one can derive a significant theoretical and practical speedup for this specific QR algorithm compared to the unmodified QR routine and the usual technique for quantum circuit synthesis based on the quantum *Shannon decomposition* (QSD) [SBM06].

From a Householder decomposition, we then propose a circuit synthesis procedure based on CNOT gates and rotations. The asymptotic counts [BBVA20, Tab.2 and Tab.3] are summarized in Table C.9. Overall, this technique turns out to be faster than the QSD-based method, although it provides circuits twice as large. One of the interest of this work is to highlight the tread-off in circuit synthesis: reducing the circuit size renders circuit generation more costly.

Circuit Synthesis with Gradient Descent. The focus of [BBVA19] is the question of the synthesis of *trapped-ions* quantum circuits. The generic structure of such circuits is a sequence

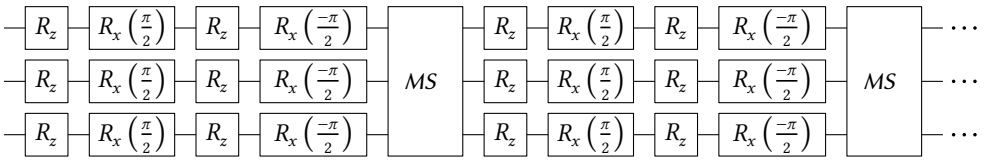


Figure C.10: Structure of trapped-ions quantum circuits

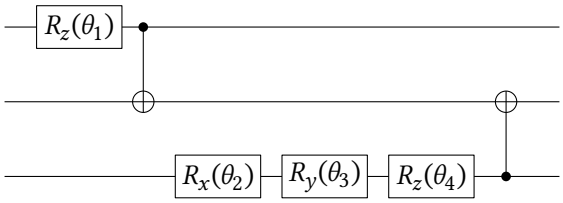


Figure C.11: Example of parameterized circuit

of layers of the form shown in Figure C.10. The gates R_z are each parameterized by a different angle, while the gates *MS* are the entangling Mølmer–Sørensen gate [MS99] defined by

$$MS(\theta) \triangleq e^{i\theta(\sum_{i=1}^{n} \sigma_X^i)^2/4},$$

with σ_X^i the operator X applied to the i-th qubit.

The question is then: for a given unitary, how many layers are needed, and, for each layer, what parameters to choose for the *MS* gate and the R_z gates? The paper [BBVA19] offers two answers: a theoretical lower bound and an experimental analysis of its optimality.

For the theoretical lower bounds, we build on a previous approach [SMB04], proposing such a lower bound in the case of circuits built from $\{SU(2), CNOT\}$. The idea is to count the number of degrees of freedom in a quantum circuit with a fixed structure and to show that this number has to exceed a certain threshold to be sure that an exact synthesis is possible for any operator.

Consider for instance the circuit shape given in Figure C.11. It can be understood as a family of circuit with at most 4 degrees of freedom (one for each rotation). In general a circuit structure with k degrees of freedom can be seen as a smooth function $f : \mathbb{R}^k \to U(2^n)$ mapping the values of angles to the space of unitary matrices of size 2^n. We are interested in the image of the function f. If for any operator U on n qubits there exists a vector of angles $\vec{\theta}$ such that $f(\vec{\theta}) = U$, then we say that the circuit shape is universal.

The contribution of [BBVA19] consists in deriving a lower bound on the number of layers required for trapped-ions circuits of the shape shown in Figure C.10, using a similar reasoning. The result is that, to be universal, a topology *must* in fact have *at least*

$$\left\lceil \frac{2^{n+1} - 2n - 2}{2n + 1} \right\rceil$$

layers of *MS* gates.

In order to address the problem of the tightness of the bound, we rely on a numerical method: The BFGS algorithm [NW06] (named after Broyden [Bro70], Fletcher [Fle70], Goldfarb [Gol70] and Shanno [Sha70]). If using heuristics or classical optimization methods to synthesize circuits had already been tried before [MMNSB16, ABIMBK19], numerical methods had however never been used to estimate the *minimum quantum resources* required to synthesize a quantum circuit.

If the paper [BBVA19] discusses the results in details, here we only want to discuss the benchmark reproduced in Figure C.12. The plot is realized as follows. One picks topologies

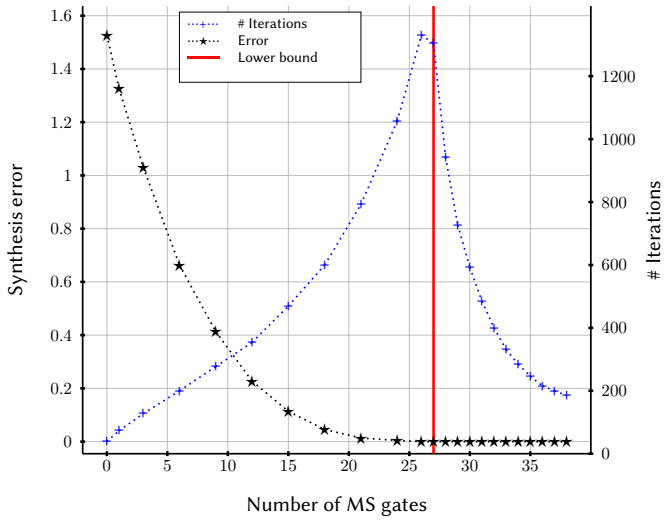


Figure C.12: 4-qubits quantum circuit synthesis problem [BBVA19].

with increasing numbers of layers. For each topology, one picks 50 random unitary matrices on 4 qubits, and the optimization process is run. The resulting plot is the obtained number of iterations and the synthesis error.

First, one can note that the error decreases exponentially with the number of MS gates: this is explained by the fact that the more layers, the larger is the number of matrices that can be reached. The interesting curve is the number of iterations required for convergence: as the number of layers increases, so does the number of required iterations for convergence, until the vertical red line. This line represents the theoretical lower bound: After this point, the number of iterations sharply decreases.

The behavior of the curve corresponding to the number of iterations is a good indication that the theoretical lower bound is indeed tight: after this point, converging becomes easier and easier as we have more degrees of freedom than necessary.

C.2.3 Circuit Synthesis from ZX Specification

Quantum —and reversible— circuits are not the only graphical language for representing quantum computation. In the late 2000's, the ZX calculus of Section B.1.5 has turned into popular alternative representation of quantum circuit [CD08, CD11, CK17]. This formal diagrammatic language with a more granular representation than quantum circuits has been successfully used in applications such as MBQC [DP10, Dun13], topological quantum computation [Hor11], Lattice-code surgery [BH20] and Pauli fusion [BDHP19], as well as for circuit simplification [DKPW20] and verification of QEC [CKRZH18] such as Steane [DL13] and Color code [GD17].

Another strength of the ZX calculus is its versatility. One can for example find variants of the ZX calculus [CJ20] such as the ZW calculus [Had15, Had17] or the ZH calculus aiming at the fragment Toffoli+H [BK18]. But the ZX calculus can also easily be equipped with extensions. One can for instance quote the SZX calculus, for reasoning on *arrays* of qubits [CHP19], graphical calculi for qudits [Ran14] and qutrits [WB14, BW15, Wan17, TM22, WY22], and the ZX$_{\perp}$ calculus, for manipulating mixed states [CJPV19].

Hybrid Quantum-Classical Synthesis from ZX$_{\perp}$ Terms. In a paper co-authored with Simon Perdrix and our Ph.D student Agustin Borgna [BPV21], we propose a circuit simplifica-

tion technique in the context of quantum and classical operations, using the ZX$_{\perp}$-calculus. This extension of the ZX-calculus adds a discarding generator—a "ground", thus the symbol "$\perp$"—to the diagrams. making it possible to represent operations interacting with the classical environment [CJPV19].

The novelty of our approach comes from the fact that common optimization strategies focus solely on the purely quantum aspect of quantum computation [AMM14, HC18]. Introduced by Duncan *et al.* [DKPW20], one of these optimization techniques uses the ZX-calculus to apply granular rewriting rules that ignore the boundaries of each quantum gate. Their rewriting steps preserve a diagram property called *gFlow* that is required for the final extraction of the ZX diagrams into circuits. Duncan's ZX optimization method was latter used by Kissinger and van de Wetering [KW20] to reduce the number of T-gates in quantum circuits.

In [BPV21], we define the natural extension of the pure Clifford optimization algorithm by Duncan *et al.* to hybrid quantum-classical circuits using the ZX$_{\perp}$ calculus. Our circuit optimization procedure forgets the difference between quantum and classical wires during the simplification process, representing connections as a single type of edge. This allows it to optimize the complete hybrid system as an homogeneous diagram, and results in similar representations for operations that can be done either quantumly or classically. We then propose a strategy to automatically recover the part of the optimized circuit that can be treated in a classical manner. Generally, in a physical quantum computer, the classical operations are simpler to implement than their quantum counterparts, and quantum simulators can exploit the knowledge of which wires carry classical data to simplify their operation. As such, it is beneficial to extract classical gates in the resulting circuit where possible.

C.3 Specification and Verification of Quantum Programs

In classical programming, a common verification technique consists in testing and debugging [Jr08]. In the case of quantum programs, this standard approach is hard to implement and bound to be insufficient [HM19a, HM19b]. A first problem is the probabilistic nature of quantum algorithms: although feasible [LZYDYX20], assertion testing is very intrusive, expensive resource-wise, and limited in its expressiveness. A second, more fundamental problem comes from the cost of running a quantum program. The cost can be monetary when running the code on a physical machine, or resource-wise when emulating it, as emulation requires an exponential quantity of classical resources. In short, we may simply not be able to afford to perform hundreds of runs of a piece of code just for testing.

If testing and debugging may not be a viable solution, a wide range of *formal verification* techniques [CW96] have been shown to be versatile tools for quantum computation, amenable to many situations. Several recent experiments have successfully adapted known formal methodologies to the quantum setting: Floyd–Hoare logics [Unr19a, Unr19b, Yin11, Yin19], use of proof-assistants [BKN15, PRZ17, RPLZ18, Ran18, HRHLH21, HRHWH21], abstract interpretation [Per08], model checking [GNP08, YLYF14, FHTZ15], equational theories [JPV18, KZ15, FD19, Amy18, Amy13, Amy19], and deductive verification [CBBPV21].

Section C.3.1 discusses the difficulties regarding formal verification of quantum programs. Section C.3.2 then presents Floyd-Hoare logic and the corresponding deductive verification techniques. Finally, Sections C.3.3 and C.3.4 presents our contribution on deductive verification of quantum programs. Section C.3.3 discusses a quantum Floyd-Hoare logic for first-order quantum programs supporting recursive calls and measurements, while Section C.3.4 presents the deductive verification framework Qbrick, based on a recent, compact semantics for quantum computation: sum-over-paths.

C.3.1 Challenges for Quantum Formal Verification

Compared to classical computation, quantum computation raises a series of problems for verification in general [CBLVVX21].

Hybrid, probabilistic model Quantum algorithms are not monolithic, linear processes: as discussed in Section C.1.1, a quantum algorithm is a subtle interaction between a classical computer and a quantum coprocessor, each having their own properties and control flow. Validating the concrete implementation of a quantum algorithm requires a suitable semantics for this hybrid model.

Furthermore, gathering classical data from the quantum memory is an inherently probabilistic procedure. In a sense, quantum computation supersedes probabilistic computation: all of the issues coming from the probabilistic settings also occur within quantum computation.

Limited resources A quantum algorithm describes *logical* quantum circuits without much care for the available resources. However, in the current NISQ era [Pre18] memory is expensive, with hardware constraints such as limited connectivity. The number of coprocessor cycles might also be limited in case of absent or limited error-correction: the evaluated circuit therefore has to be kept under a certain depth [CBSNG19]. In the current state of the technique, adapting algorithms to the noise constraints can be challenging [GE21]. This makes quantum coprocessors akin to embedded systems: there is a need for a fine-grained resource management. If programming languages such as Quipper [GLRSV13b] can help with resource estimation, dedicated compilation tools have been developed to automate circuit optimization and physical qubit layout [AMM14, PRS17, MSRH20, HRHWH21, SDCSED20].

Functional specifications A quantum algorithm comes with a *functional specification* describing its behavior. There are two kinds of functional specifications. On one hand, an *intentional* specification considers the algorithm as an opaque instantiation of a mathematical function, and only discusses the relationship between the input and the output of the algorithm. Ying's quantum Floyd–Hoare logic [LZWY+19a, Yin11] is a typical approach leaning towards intentional presentations. On the other hand, an *extentional* specification "opens the box" and also describes *how* the computation gets to the result. An extentional specification might then for instance give requirements on the size and shape of a circuit produced by the algorithm. Approaches for extentional presentations typically use dependent type systems [PPZ19] or embeds into program verification tools such as Coq [HRHWH21, PRZ17] or Why3 [FP13, CBBPV21].

Compilation Toolchain As discussed in Section C.1.1, a quantum program is not only the description of *one* quantum circuit: at a minimum it describes a *family* of quantum circuits, parameterized by the problem instance. A specification concerns this family —a versatile verification tool should be able to handle parametricity.

Furthermore, along the compilation process the generated (families of) circuits are transformed and optimized according to various constraints coming from the error model, the hardware connectivity, the cost of each gates, etc. In general, these transformations also need to be validated: they should not modify the *semantics* of the circuit. This semantics in general involves linear algebra: the validation tools should therefore handle it [HRHWH21, CBBPV21], or restrict to subsets such as reversible circuits [PRS17].

C.3.2 Floyd–Hoare Logic and Deductive Verification

Deductive program verification is probably the oldest formal method technique, dating back to Floyd and Hoare in the 1960's [Flo67, Hoa69]. In this approach —the so-called *Floyd–Hoare logic*— a piece of code C is annotated with a logical contract [Mey92] consisting of a pre-condition P and a post-condition Q. The tuple $\{P\}C\{Q\}$ is valid if whenever P is true, executing the code C makes the post-condition Q valid.

If one of the parameters of a Floyd–Hoare logic is the programming language, the other parameter is the chosen logic. The objective is to allow for a logic as expressive as possible, while being able to give complete set of syntactic deduction rules together with an algorithm as efficient as possible for inferring a valid proof of a contract. The seminal works underlying the whole development of the field are Dijkstra's algorithm for *weakest precondition* inference [Dij76] and Burstall's proposal for the addition of intermittent assertions [Bur74].

The weakest precondition inference algorithm is at the core of the automation permitted by deductive verification based on Floyd–Hoare logic. Consider the rule for sequential composition:

$$\frac{\{P\}C_1\{Q\} \quad \{Q\}C_2\{R\}}{\{P\}C_1; C_2\{R\}.} \tag{C.5}$$

It states that for R to be a valid-postcondition for the program $C_1; C_2$ under the pre-condition P, one simply has to find Q that is both pre-condition for C_2 and post-condition for C_1. Dijkstra's algorithm automates the discovery of a most-general pre-condition $\mathsf{wp}(F, C)$ for a code C with post-condition F. Rule (C.5) dictates that one can pick Q to be $\mathsf{wp}(R, C_2)$. The weakest pre-condition for $C_1; C_2$ then becomes $\mathsf{wp}(\mathsf{wp}(Q, C_2), C_1)$. To recover $\{P\}C_1; C_2\{R\}$, a *proof obligation* is generated:

$$\mathsf{wp}(\mathsf{wp}(Q, C_2), C_1) \Rightarrow P.$$

This formula can either be proven in a proof assistant or discharged with an SMT-solver.

In the context of classical programming, these techniques have been applied in academic or industrial contexts for many languages [HH19]. One can cite frameworks for the Pascal language [DHKK95], Ada [LH85, SPA11], Modula-3 [LN98], Java [HAGH16, FM07, RLNS00], C [Nor98, FM07, KKPSY15], the Method B capitalizing on Dijkstra's weakest precondition algorithm [Rob97, LFFP11], and the versatile Why3 environment for WhyML [FP13].

C.3.3 Quantum Floyd–Hoare Logic Handling Measurements

Two main Floyd–Hoare logics specific to quantum computation have emerged in recent years. The first line of work [Unr19b, Unr19a, BHYYZ20] proposes a Floyd–Hoare logic for reasoning about programs implementing quantum protocols. The framework is based on regular, classical logical constructors. The logic is extended with the capability to reason about variables holding quantum states, such as "x holds the qubit state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$".

The second approach to quantum Floyd–Hoare logic is now more than 10 years old and stems from Ying's research group [Yin11, YYFD13, YYW17, LWZG+18, LY18, Yin19, ZYY19, LZWY+19a, HHZYHW19, LZWY+19b, BHYYZ20, LZBY22, FLY22, YZLF22]. This prolific research avenue can be traced back to d'Hondt and Panangaden's work on *quantum weakest precondition* [dP06, dP04], quantum equivalent to Dijkstra's notion. D'Hondt and Panangaden's idea consists in regarding positive operators as (probabilistic) formulas on states. Remember that the probability of measuring the density matrix ρ in state $|\phi\rangle$ is $\langle\phi|\,\rho\,|\phi\rangle$. This can be rewritten as $\mathrm{Tr}(M\rho)$, with $M = |\phi\rangle\langle\phi|$. In general, M can be any *observable* operator, which for our purpose we can consider as a density matrix. For instance, the observable $\frac{1}{3}|0\rangle\langle 0| + \frac{2}{3}|1\rangle\langle 1|$ assesses the probability of ρ to be in the mixed state $\frac{1}{3}\{|0\rangle\} + \frac{2}{3}\{|1\rangle\}$.

Quantum programs in Ying's approach are the quantum equivalent of a textbook while-language: a fixed set of possible variables, all quantum, and each spanning a given Hilbert space, and a few imperative constructs with sequential composition for acting on the state of the variables: assertion, tests, while-loop. Since the only available types are quantum, branching is probabilistic and based on the result of a measurement.

One can for instance write the program

$$x := H\,x; \texttt{while}\ (|0\rangle\langle 0|x = |0\rangle)\ \{x := H\,x\} \tag{C.6}$$

which repeatedly measure x against $|+\rangle$ until $|1\rangle$ is obtained. An observable serving as formula is in this case acting on the state of x (i.e. it is acting on a qubit).

Given a post-condition Q and a program C, a *quantum precondition* is an operator P such that for any density matrix ρ representing a state of the memory of C, $\mathrm{Tr}(P\,\rho) \leq \mathrm{Tr}(Q\,(C\,\rho))$, with $C\,\rho$ the state of the system after the action of C. The operator P is *weakest precondition* for Q and P if for all other precondition P', we have $P' \sqsubseteq P$ (using the Löwner order). For instance, a precondition for the program shown in Eq (C.6) and the postcondition $|1\rangle\langle 1|$ is $|0\rangle\langle 0|$. It is of course not unique —the operator 0 is also a pre-condition— but it is the weakest one.

Minsheng Ying's group has extensively worked on this approach, with special attention to the structure of invariants required for the while-loop. The group studied various extensions and problems such as non determinism [LYY14], testing and debugging [LZYDYX20], linear-time properties [YLYF14], termination and expected run-time [LY18, LZBY22], parallel and distributed quantum programs [FLY22, YZLF22].

At a high-level, if Ying's approach makes it easy to discuss probabilistic behavior (since it is *part of* the structure of the logic), the shallow embedding inside operators —similar to Birkhoff and von Neumann's quantum logics [BN36, Mit78]— limits its expressiveness. For instance, It is hardly extensible to features such as first-order or native manipulation of conventional types such as natural numbers, lists, *etc.*

> In the context of the ANR project SoftQPro, I had the opportunity to dig into the subject with a former collaborator of Minsheng Ying, Zhaowei Xu, hired as a postdoc in our group. We worked on an extension of Ying's approach to quantum Hoare logic, to allow local variables and recursive subroutines. This collaboration yielded a paper to appear in TOCL [XVY21].

C.3.4 Qbrick: Deductive Verification with Parametrized Path Sums

> In 2017, I was invited at CEA-LIST/LSL by François Bobot and Sébastien Bardin to give a seminar to present Quipper. Along the discussion afterwards, we came to the conclusion that Why3 [FP13] could very well serve as a host language for a quantum programming language, and that it could freely provide a means to certify and verify embedded quantum programs. Unlike Ying's quantum Hoare logic, the Why3 logic seemed expressive enough to state both intentional and extentional properties of programs.
> The project effectively started when they hired Christophe Chareton as postdoc to build on the idea. Moving from an hypothetical concept to a concrete tool able to prove Shor's algorithm properties took about 3 years. In [CBBPV21], we present the outcome: Qbrick, a DSL embedded in Why3 coming with dedicated libraries of definitions and lemmas based on sum-over-paths [Amy19], dedicated to the formalization of quantum programs in a deductive verification framework. If I was involved in the theoretical development behind Qbrick, Christophe has been the kingpin of the development of the toolbox.

In the situation described in Chapter C, a quantum program might manipulate quantum registers of large dimension. For specification and verification purposes, this renders the technique presented in Section C.3.3 hard to use: not only proofs become sprawling but also positive operators in logical formulae becomes daunting. This renders pen-and-paper proofs impossible.

One solution consists in relying on a proof-assistant and to code intentional properties inside the corresponding logic. This has been done in Isabelle/HOL for Ying's Hoare logic [LZWY+19a], and in Coq for the QWIRE language [PRZ17, Ran18], followed by the VOQC

framework [HRHLH21]. However, these approaches relies on (concrete) matrices, whether unitary [PRZ17] or positive [LZWY+19a]. As it turns out, matrices are not well-suited for automation, and long, manual proofs are necessary for validating formal specification of quantum programs in this formalism.

In the paper [CBBPV21], we propose an alternative solution. On one hand, instead of using a generic proof-assistant such as Coq [PRZ17, Ran18] or Isabelle/HOL [LZWY+19a, MAT14], we rely on Why3 [FP13], a platform for deductive verification dedicated to proof automation [BFMP11]. On the other hand, instead of using the hard-to-automate matrix formalism, we rely on a compositional, functional semantics: *sum-over-paths* (or path-sums) [Amy18, Amy19].

Sum-over-Paths. Amy's *path-sum* semantics offer an algebraic, intentional presentation of quantum circuits, alternative to the matrix presentation. The name comes from the correspondence with Feynman's path integral [FH65]. This very versatile framework is amenable to other formalisms of quantum computation such as ZX calculus [CK17, Vil21].

The idea consists in formalizing the standard function-style presentation of an operator A

$$|x\rangle \quad \longmapsto \quad \sum_{k=0}^{2^n} \alpha_{k,x} |k\rangle \,.$$

Instead of listing all of the $\alpha_{k,x}$'s exhaustively, the operator is written as a triple (m, P, ϕ), where m is an integer and P and ϕ are integer polynomials such that A is

$$|x\rangle \quad \longmapsto \quad \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2i\pi \cdot P_k(x)}{2^m}} |\phi_k(x)\rangle \,.$$

For many "interesting" operators, the polynomials P and ϕ form a more compact representation than the array of the $\alpha_{k,x}$'s. Furthermore, this representation is closed under functional composition and Kronecker product, making it ideal for reasoning on quantum circuits.

One limitation of Amy's path sum is however that one cannot check *parameterized family* of circuits: akin to a model-checker, the path-sum mechanism can only handle one *fixed* circuit.

The Domain-Specific Language Qbricks In [CBBPV21] we propose Qbricks, a specification and verification framework for quantum programs based on path-sums. Before Qbricks, frameworks for proving properties of quantum programs where either handling parametricity at the expense of automation [MAT14, LZWY+19a, PRZ17, RPLZ18, HRHLH21] or automated at the expense of parametricity [Amy18, Amy19, KZ15]. Our contribution to the field consists in reconciliating parametricity and automation, with the development of a deductive verification framework based on *parameterized path sums*.

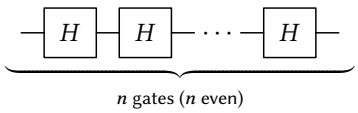
Qbricks is embedded in Why3, inheriting its specification and deductive verification features. The formalization comes with a domain-specific language for circuit manipulation and a logic library for manipulating path-sums. This gives a handle for reasoning in terms of the WhyML language: our path-sums are naturally parameterized.

Qbricks' domain specific language is following the qPCF's strategy for circuit construction [PZ17] —although qPCF is mainly a theoretical exploration of dependent type systems in this context. Unlike Quipper where wires are qubits that can be instantiated and manipulated as variable and where circuits are functions on qubits, circuits in Qbricks are opaque objects manipulated with a few combinators: elementary gates, sequential and parallel composition.

The framework has been used to prove the first verified, parametric implementation of the quantum part of Shor's factoring algorithm [Sho94, Bea03], including both the polynomial complexity of the circuits and the probability requirements. We also experimented with Grover [Gro96] and the quantum phase estimation subroutine (QPE) [Kit95]. Our method [CBBPV21,

Sec 8] achieves a high level of proof automation (96% on Shor) and significantly reduces proof effort (factor 13.6x compared with [LZWY+19a] on Grover, factors 7.7x and 6.4x compared with [HRHLH21] on respectively QPE and Grover).

Example of Parametric Path-Sums Let us present an example to illustrate the interplay between the language and the parametric path-sums. Consider the family of circuits defined as an even number of Hadamard gates



We can give a specification for a program generating such a circuit family by

Precondition $n \geq 0$ is even

Postcondition C_n sends $|x\rangle$ to $|x\rangle$ and C_n consists of n gates.

This contrived example is typical for the specification of a quantum algorithm:

- the description of the circuit family is parameterized by a classical parameter (here, the non-negative integer n);
- The precondition imposes both constraints (here, the evenness of n) and soundness conditions (here, the non-negativeness of n) on the parameters;
- The postcondition can both refer to the semantics of the circuit result and to its form and shape (here, its size).

Regular path-sums are not adequate for representing the semantics of the circuit family since the behavior of each circuit in the family depends on its size: the path sum is

$$|x\rangle \mapsto \begin{cases} \frac{1}{\sqrt{2^0}} \sum_{k=0}^{2^0-1} e^{2i\pi\cdot 0} |x\rangle & \text{when } n \text{ is even} \\ \frac{1}{\sqrt{2^1}} \sum_{k=0}^{2^1-1} e^{2i\pi\cdot \frac{kx}{2}} |k\rangle & \text{when } n \text{ is odd.} \end{cases}$$

Compared to Amy's proposal, the phase and boolean polynomials of path-sums are generalized to generic, parameterized terms. In the case of our example, the path-sum becomes

$$|x\rangle \mapsto \frac{1}{\sqrt{2^{n\%2}}} \sum_{k=0}^{2^{n\%2}-1} e^{2i\pi\cdot \frac{(n\%2)kx}{2}} |\texttt{if even}(x)\ \texttt{then}\ x\ \texttt{else}\ k\rangle\,.$$

With Qbricks' framework, such a path-sum can be defined in the language and reasoned upon in the logic.

[GLRSV13b] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. "Quipper: a scalable quantum programming language". In: *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'13* (Seattle, WA, USA). Ed. by Hans-Juergen Boehm and Cormac Flanagan. ACM, 2013, pp. 333–342. ISBN: 978-1-4503-2014-6. DOI: 10.1145/2491956.2462177. arXiv: 1304.3390.

[GLRSV13a] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. "An introduction to quantum programming in quipper". In: [RC13], pp. 110–124. DOI: 10.1007/978-3-642-38986-3_10. arXiv: 1304.5485.

[SRSV14] Jonathan M. Smith, Neil J. Ross, Peter Selinger, and Benoît Valiron. "Quipper: concrete resource estimation in quantum algorithms". In: *Informal Proceedings of QAPL'14, Grenoble, France*. 2014. arXiv: 1412.0625.

[VRSAS15] Benoît Valiron, Neil J. Ross, Peter Selinger, Dana Scott Alexander, and Jonathan M. Smith. "Programming the quantum future". In: *Communications of the ACM* 58.8 (2015), pp. 52–61. DOI: 10.1145/2699415. URL: http://doi.acm.org/10.1145/2699415. HAL: hal-01194416.

[Val16] Benoît Valiron. "Generating reversible circuits from higher-order functional programs". In: *Proceedings of the 8th International Conference on Reversible Computation, RC'16* (Bologna, Italy). Ed. by Simon J. Devitt and Ivan Lanese. Vol. 9720. Lecture Notes in Computer Science. Springer, 2016, pp. 289–306. DOI: 10.1007/978-3-319-40578-0_21. HAL: hal-01474621.

[Val17] Benoît Valiron. *Programmer un ordinateur quantique*. Column in MathsInfos Hors-Série Numéro 3, published by Fondation Mathématique de Paris. 2017. HAL: hal-01763585.

[SVMABC17] Artur Scherer, Benoît Valiron, Siun-Chuon Mau, D. Scott Alexander, Eric van den Berg, and Thomas E. Chapuran. "Concrete resource analysis of the quantum linear-system algorithm used to compute the electromagnetic scattering cross section of a 2D target". In: *Quantum Information Processing* 16.3 (2017), p. 60. DOI: 10.1007/s11128-016-1495-5. HAL: hal-01474610. arXiv: 1505.06552.

[ABGV18] C. Allouche, M. Baboulin, T. Goubault de Brugière, and B. Valiron. "Reuse method for quantum circuit synthesis". In: *Recent Advances in Mathematical and Statistical Methods, post-proceedings of the IV AMMCS International Conference on Applied Mathematics, Modeling and Computational Science, Waterloo, Canada, August 20 – 25, 2017*. Ed. by D. Marc Kilgour, Herb Kunze, Roman Makarov, Roderick Melnik, and Xu Wang. Springer International Publishing, 2018, pp. 3–12. ISBN: 978-3-319-99719-3. DOI: 10.1007/978-3-319-99719-3_1. HAL: hal-01711378.

[Val18] Benoît Valiron. "A formal analysis of quantum algorithms". In: *ERCIM News* 112 (Jan. 2018), pp. 23–24. HAL: hal-01763602.

[BBVA19] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, and Cyril Allouche. "Synthesizing quantum circuits via numerical optimization". In: *Proceedings of the 19th International Conference on Computational Science, ICCS 2019, Part II* (Faro, Portugal, June 12–14, 2019). Ed. by João M. F. Rodrigues, Pedro J. S. Cardoso, Jânio M. Monteiro, Roberto Lam, Valeria V. Krzhizhanovskaya, Michael Harold Lees, Jack J. Dongarra, and Peter M. A. Sloot. Vol. 11537. Lecture Notes in Computer Science. Springer, 2019, pp. 3–16. ISBN: 978-3-030-22740-1. DOI: 10.1007/978-3-030-22741-8_1. HAL: hal-02174967. arXiv: 2004.07714.

[BBVMA20] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Quantum CNOT circuits synthesis for NISQ architectures using the syndrome decoding problem". In: [RC20], pp. 189–205. DOI: 10.1007/978-3-030-52482-1_11.

[BBVA20] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, and Cyril Allouche. "Quantum circuits synthesis using Householder transformations". In: *Computer Physics Communications* 248 (2020), p. 107001. DOI: 10.1016/j.cpc.2019.107001. HAL: hal-02545123. arXiv: 2004.07710.

[CBBPV21] Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. "An automated deductive verification framework for circuit-building quantum programs". In: *Proceedings of the 30th European Symposium on Programming Languages and Systems, ESOP 2021* (Luxembourg City, Luxembourg, Mar. 27–Apr. 1, 2021). Ed. by Nobuko Yoshida. Vol. 12648. Lecture Notes in Computer Science. Springer, 2021, pp. 148–177. ISBN: 978-3-030-72018-6. DOI: 10.1007/978-3-030-72019-3_6. arXiv: 2003.05841.

[BPV21] Agustín Borgna, Simon Perdrix, and Benoît Valiron. "Hybrid quantum-classical circuit simplification with the ZX-calculus". In: *Proceedings of the 19th Asian Symposium on Programming Languages and Systems, APLAS 2021* (Chicago, IL, USA (Online Conference), Oct. 17–18, 2021). Ed. by Hakjoo Oh. Vol. 13008. Lecture Notes in Computer Science. Springer, 2021, pp. 121–139. DOI: 10.1007/978-3-030-89051-3_8. arXiv: 2109.06071.

[CBLVVX21] Christophe Chareton, Sébastien Bardin, Dongho Lee, Benoît Valiron, Renaud Vilmart, and Zhaowei Xu. "Formal Methods for Quantum Programs: A Survey". Draft, to appear as a book chapter. 2021. arXiv: 2109.06493.

Table C.13: Personal publications related to Chapter C.

Chapter D

Semantics of Quantum Lambda-Calculi

Semantics can be considered the origin of all the formal tools developed to analyze, certify, and verify programming languages [Lee90]. It consists in a formal description of the essence of programs aiming at unearthing the structures underlying the capabilities of programming language. Semantics draws links between the behavior of programs—how they evolve and interact with their environment—, their logical properties—how they are structured—, and the result of their action—what they compute.

For classical, regular programming languages, semantics—and formal methods—have been around for more than half a century. Based on powerful frameworks such as category theory or the Curry-Howard isomorphism [CFC58, How80], semantics for classical programming languages gave birth to a range of fine-grained analysis techniques of programs.

For classical programming languages, the underlying mathematical structures are typically set-based, discrete structures [Sto77]. Although the analysis of quantum programming languages can rely on and adapt some of the work done in the classical setting, several aspects fundamentally differ and require novel techniques. In particular, in quantum computing, one deals with two kinds of objects: regular, duplicable objects and quantum, non-duplicable objects. Moreover, the canonical mathematical representation of quantum states is based on vector spaces and operator algebras.

Developing a semantics for a quantum programming language then requires a novel approach. In this chapter, we present our contribution to the field, focusing on the quantum lambda calculus and its extension as a circuit-description language.

- Section D.1 summarizes the base of our main approach: the fact that linear logic forms a suitable framework for a quantum type system, following Section B.3.3.
- Section D.2 discusses the procedure we followed for building a denotational semantics accounting for both quantum *and* duplicable data. This kind of semantics interprets programs as functions. The semantics we propose is strongly inspired by quantitative semantics of linear logics [PSV14].
- Section D.3 focuses on a complementary approach: the Geometry of Interaction. This technique provides executable semantics based on token-based automata. We show how quantum lambda-terms can be regarded as folded quantum circuits; the semantics gives a procedure for "running" them [LFVY15, LFVY17].
- Finally, Section D.4 briefly discusses one of our recent results: a categorical semantics for ProtoQuipper, a circuit-description extension of the quantum lambda calculus sup-

porting dynamic lifting: the ability to govern circuit generation based on the result of previous measurements [LPVX21].

D.1 Linear Logic and Typed Quantum Lambda Calculus

As discussed in Section B.3.3, a natural logical framework for a type system for quantum computation is linear logic. In this section, we briefly introduce the logic and how it lays out a natural type system for the quantum lambda calculus.

D.1.1 Linear Logic

The logic formula from *linear logic* (LL) that we shall be considering are

$$A, B ::= \alpha \mid A^\perp \mid \mathbf{1} \mid \bot \mid \mathbf{0} \mid \top \mid A \otimes B \mid A \parr B \mid A \oplus B \mid A \& B \mid !A \mid ?A,$$

where α ranges over a set of atomic formulas. In linear logic there are two pairs of conjunctions/disjunctions: a multiplicative version: $\otimes$ (with unit $\mathbf{1}$) and $\parr$ (with unit $\bot$), and an additive version: $\&$ (with unit $\top$) and $\oplus$ (with unit $\mathbf{0}$). The connective $(-)^\perp$ stands for the linear negation. It is extended to an involution on formulas where $(A^\perp)^\perp = A$ as follows:

$$\begin{aligned} \mathbf{1}^\perp &= \bot & (A \otimes B)^\perp &= A^\perp \parr B^\perp, & (!A)^\perp &= ?(A^\perp), \\ \mathbf{0}^\perp &= \top & (A \oplus B)^\perp &= A^\perp \& B^\perp, \end{aligned}$$

emphasizing the fact that $\otimes/\parr$, $\oplus/\&$, $!(-)/?(-)$, $\mathbf{1}/\bot$ and $\mathbf{0}/\top$ are *dual* connectors. Intuitively, a negated formula stands for an *hypothesis* (i.e. an "input") while a non-negated formula for a *conclusion* (i.e. an "output"). Following the intuition that $\parr$ is a disjunction, we define a macro $A \multimap B = A^\perp \parr B$, then representing a (multiplicative) linear implication. In light of the duality of connectives, we can give a meaning to the two last connectives, the modalities $!(-)$ (the *exponential*) and $?(-)$. Indeed, in linear logic, formulas are *linear* by default: they correspond to resources that have to be used exactly once. The connectives $!(-)$ and $?(-)$ make it possible to relax this constraint: $!A$ stands for a duplicable and erasable *output* of type A, while $?A$ stands for a duplicable and erasable *input* of type A.

Example D.1. According to the intuitive meaning we gave to the linear logic connectives, without additional axioms the formula $\alpha \multimap \alpha$ should then be correct, while $\alpha \multimap (\alpha \otimes \alpha)$ should not. On the other hand, $!\alpha \multimap (\alpha \otimes \alpha)$ should be valid, since $!A$ is a "duplicable" resource.

As in classical logic, linear logic features a notion of *sequent*, that is, a sequence of formulas, denoted with $\vdash A_1, \ldots, A_n$. We call "$\vdash$" a *turnstyle*. Generic sequences of formulas are denoted with $\Delta, \Gamma, \ldots$. If $\Delta = x_1 : A_1, \ldots, x_n : A_n$, and if $\square$ is a unary connective, we write $\square\Delta$ for the sequence $x_1 : \square A_1, \ldots, x_n : \square A_n$.

We say that a sequent is *valid* if it can be derived from the following rules.

$$\frac{}{\vdash \mathbf{1}}\ (\mathbf{1}) \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\ (\bot) \qquad \frac{}{\vdash \Gamma, \top}\ (\top) \qquad \frac{}{\vdash A, A^\perp}\ (\text{ax}) \qquad \frac{\vdash A_1, \ldots A_n}{\vdash A_{\sigma(1)}, \ldots, A_{\sigma(n)}}\ (\text{ex}_\sigma)$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}\ (\text{cut}) \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\ (\otimes) \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ (\parr)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\ (\oplus_1) \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\ (\oplus_2) \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B}\ (\&)$$

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\ (\text{p}) \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ (\text{d}) \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\ (\text{w}) \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\ (\text{c})$$

where σ is a permutation over $\{1, \dots n\}$. Note that there is no rule for the unit $\mathbf{0}$. By abuse of notation the rule (ex_σ) is left implicit in the description of proofs. From the rules one can check that $\otimes$ indeed behaves like a conjunction while $\parr$ behaves like a disjunction. One can also see how $\otimes/\&$ has a multiplicative flavor —contexts are disjoints— while $\oplus/\&$ has an additive flavor —contexts are shared—.

Remark D.2. Note that the position of the formulas in a sequent is essential, as otherwise the following proof is ambiguous:

$$\dfrac{\dfrac{}{\vdash A, A^\perp}\ (\mathrm{ax}) \quad \dfrac{}{\vdash A, A^\perp}\ (\mathrm{ax})}{\vdash A, A^\perp}\ (\mathrm{cut}). \tag{D.1}$$

Which pair $A, A^\perp$ was canceled out by the (cut)-rule?

Remark D.3. Two derivable rules are often added; they are specially useful when considering proof-nets.

$$\dfrac{}{\vdash}\ (\mathrm{empty}) \qquad \dfrac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta}\ (\mathrm{mix})$$

in which case we refer to the logic as LL+mix.

Example D.4. A linear Modus-Ponens can be derived as follows, where we add a dummy rule for highlighting the unfolding of $\multimap$:

$$\dfrac{\begin{array}{c}\vdots\ \pi_1 \\ \vdash A\end{array} \qquad \dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\ (\mathrm{ax}) \quad \dfrac{}{\vdash B, B^\perp}\ (\mathrm{ax})}{\vdash A \otimes B^\perp, B, A^\perp}\ (\otimes)}{\vdash (A \multimap B)^\perp, B, A^\perp}\ (\mathrm{unfold}) \quad \begin{array}{c}\vdots\ \pi_2 \\ \vdash A \multimap B\end{array}}{\vdash B, A^\perp}\ (\mathrm{cut})}{\vdash B}\ (\mathrm{cut}) \tag{D.2}$$

In the proof of Eq. (D.2), we omitted a call to the rule (ex_σ) at the (unfold) position: a full proof with sequents is potentially verbose with many bureaucratic permutations of formulas.

Remark D.5. The turnstyle notation for sequent can be extended by identifying $\Delta \vdash \Gamma$ and $\vdash \Delta^\perp, \Gamma$. The notation adds the meta-information that Δ is to be regarded as an input and Γ as an output. This triggers one interesting variant of linear logic for this chapter: *intuitionistic linear logic* (ILL) [Tro92]. In ILL, we consider special sequents with exactly one formula as conclusion: sequents are of the form $\Delta \vdash A$. Additionally, the negation $(-)^\perp$ is not anymore an involution.

Intuitionistic logic can be faithfully encoded inside linear logic [Gir87, Sec 5.1]. Regular, classical simply-typed programs can therefore be mapped to proofs of linear logics; cut-elimination then corresponds to program evaluation. Passing through a linear-logic encoding gives a fine-grained handle on the choice of evaluation strategy through the placement of the exponential modality [Sim05]. The intuition is to consider a term typed with $!A$ as a *thunk*: a frozen computation. It can be duplicated (with contraction), erased (with weakening), and run with dereliction. Historically, there are two canonical encodings building on this intuition: one implementing call-by-value, where $- \to -$ is mapped to $!(- \multimap -)$ the other one call-by-name, where $- \to -$ is mapped to $(!-) \multimap B$.

We conclude this section by mentioning interesting *fragments* of linear logic, each one with a intuitionistic and a classical variant. The first one can be inferred from Example D.4: *Multiplicative Linear Logic* (MLL), where formulas are restricted to $\otimes$ and $\parr$ (and $\multimap$). This logical fragment is *purely linear*. There is then *Multiplicative Exponential Linear Logic* (MELL), where formulas consists of $\otimes$, $\parr$ together with the *modalities* "!" and "?". These are the two fragments that we shall be considering in this paper. We can nonetheless mention the (strictly linear) fragment MALL of *Multiplicative, Additive Linear Logic* with $\otimes/\parr$ and $\oplus/\&$.

D.1.2 Quantum Lambda Calculus and Linear Logic

We claimed in B.3.3 that linear logic forms a natural framework for a type system of quantum lambda calculi. In this section, we present the instantiation described in [PSV14]: it will serve as a support for the rest of the discussion in this chapter.

The language is defined as follows.

$$M, N, P ::= x \mid \lambda x.M \mid MN \mid \tag{D.3}$$

$$\langle M, N\rangle \mid \mathtt{let}\ \langle x, y\rangle = M\ \mathtt{in}\ N \mid \langle\rangle \mid \mathtt{let}\ \langle\rangle = M\ \mathtt{in}\ N \mid \tag{D.4}$$

$$\mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ P \mid \tag{D.5}$$

$$U \mid \mathtt{qinit} \mid \mathtt{meas} \mid \tag{D.6}$$

It consists of a regular lambda calculus (D.3), extended with: pairing constructs (D.4), where $\langle M, N\rangle$ stands for the pair of M and N and $\langle\rangle$ is the unit-term; Boolean values and tests (D.5); constants for manipulating qubits (D.6), where U ranges over a fixed set of unitary maps. The language can also be extended with recursion, using the following construct:

$$\mathtt{let\,rec}\ f\ x = M\ \mathtt{in}\ N. \tag{D.7}$$

The type system for the language is as follows.

$$A, B ::= \mathtt{qbit} \mid \mathtt{bit} \mid A \multimap B \mid A \otimes B \mid \mathbf{1} \mid !A. \tag{D.8}$$

It consists of two constant types `qbit`, for representing qubits, and `bit`, for the Boolean values $\mathtt{tt}$ and $\mathtt{ff}$, and type constructors: for pairing ($A \otimes B$), functions ($A \multimap B$), unit-type $\mathbf{1}$ for representing $\langle\rangle$, and duplicable elements ($!A$). We use the same notations as MELL to highlight the relationship with the logic. The tensor is associative to the right: $A \otimes B \otimes C = A \otimes (B \otimes C)$. We write $A^{\otimes n}$ for the n-th tensor of A. Finally, we define a notion of *value*:

$$V, W \quad ::= \quad x \mid \mathtt{tt} \mid \mathtt{ff} \mid \lambda x.M \mid \langle\rangle \mid \langle V, W\rangle.$$

We consider terms to be implicitly typed: every subterm comes with a type. A *typing judgment* is a triple written $\Delta \vdash M : A$, where M is a (typed) term, A is a type and Δ is an unordered list of typed variables: $\Delta = x_1 : A_1, \ldots, x_n : A_n$. A typing judgment is *valid* if it can be derived from the typing rules presented below. We also require that whenever $\Delta \vdash M : A$ is valid, then A is the implicit type of M.

The typing rules follow the proof rules of *intuitionistic* linear logic (as discussed in Remark D.5). The typing rules for the core lambda calculus of (D.3) are as follows.

$$\frac{}{!\Delta, x : A \vdash x : A}\ (\mathrm{ax}) \qquad \frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x.M : A \multimap B}\ (\multimap_I)$$

$$\frac{!\Delta, \Gamma_1 \vdash M : A \multimap B \quad !\Delta, \Gamma_2 \vdash N : A}{!\Delta, \Gamma_1, \Gamma_2 \vdash MN : B}\ (\multimap_E)$$

Note how contraction is included inside the $(\multimap_E)$-rule: this will be the case for every branching rule. Also note how a cut-rule is implicitly used, as it will be the case for every elimination rules.

The pairing constructs correspond to the proof rules of $\otimes$ and $\mathbf{1}$.

$$\frac{!\Delta, \Gamma_1 \vdash M : A \quad !\Delta, \Gamma_2 \vdash N : B}{!\Delta, \Gamma_1, \Gamma_2 \vdash \langle M, N\rangle : A \otimes B}\ (\otimes_I) \qquad \frac{}{!\Delta \vdash \langle\rangle : \mathbf{1}}\ (\mathbf{1}_I)$$

$$\frac{!\Delta, \Gamma_1 \vdash M : A \otimes B \quad !\Delta, \Gamma_2, x : A, y : B \vdash N : C}{!\Delta, \Gamma_1, \Gamma_2 \vdash \mathtt{let}\ \langle x, y\rangle = M\ \mathtt{in}\ N : C}\ (\otimes_E)$$

$$\frac{!\Delta, \Gamma_1 \vdash M : \mathbf{1} \quad !\Delta, \Gamma_2 \vdash N : C}{!\Delta, \Gamma_1, \Gamma_2 \vdash \mathtt{let}\ \langle\rangle = M\ \mathtt{in}\ N : C}\ (\mathbf{1}_E)$$

If contraction is included inside branching rules, weakening is handled at axiom rules. The two remaining rules for manipulating modalities —*dereliction* and *promotion*— each feature an explicit rule but with a caveat. We force dereliction to only happen at a leaf of the typing derivation, in order to ensure uniqueness of typing derivation. For technical convenience, we also restrict duplication to function-types. Finally, promotion is constrained to *values* this is in line with the call-by-value operational semantics of the languages, presented below.

$$\frac{}{!\Delta, x : !(A \multimap B) \vdash x : A \multimap B}\ (\text{axd}) \qquad \frac{!\Delta \vdash V : A \multimap B \quad V \text{a value}}{!\Delta \vdash V : !(A \multimap B)}\ (\text{p})$$

If the language can be expanded with additive types —and even inductive types— [PSV14], in this chapter we only consider Boolean values, which, albeit weaker, already capture much of the intricacy of the additives.

$$\frac{}{!\Delta \vdash \mathtt{tt}, \mathtt{ff} : \mathtt{bit}}\ (\mathtt{tt}, \mathtt{ff}) \qquad \frac{!\Delta, \Gamma_1 \vdash P : \mathtt{bit} \quad !\Delta, \Gamma_2 \vdash M, N : C}{!\Delta, \Gamma_1, \Gamma_2 \vdash \mathtt{if}\ P\ \mathtt{then}\ M\ \mathtt{else}\ N : C}\ (\text{if})$$

Finally, the constants for manipulating qubits are typed as follows.

$$\frac{}{!\Delta \vdash \mathtt{qinit} : \mathtt{qbit} \multimap \mathtt{bit}} \qquad \frac{}{!\Delta \vdash \mathtt{meas} : \mathtt{bit} \multimap \mathtt{qbit}} \qquad \frac{}{!\Delta \vdash U : \mathtt{qbit}^{\otimes n} \multimap \mathtt{qbit}^{\otimes n}}$$

In the rule for U, the number n stands for the arity of U.

Remark D.6. Note how there is no need for an exchange rule similar to (ex_σ), since types are indexed with variables. This is one of the solutions to the problem of bureaucracy; the other one is to use proof-nets, discussed in Section D.3.

The language is equipped with an operational semantics in the form of the abstract machine described in Section B.3.2: a program is a triple $[Q, L, M]$, where $Q \in \boldsymbol{Q}^{\otimes n}$ is a normalized vector of dimension 2^n, M is a term with n free variables $x_1, \ldots, x_n$ and L is a bijection $\{x_1, \ldots, x_n\} \to \{1, \ldots, n\}$. The variables x_i represent qubits inside the term M. A program is well-typed of type A, written $[Q, L, M] : A$, whenever

$$x_1 : \mathtt{qbit}, \ldots, x_n : \mathtt{qbit} \vdash M : A$$

is valid.

Programs are equipped with a *probabilistic* rewrite system ($\to_p$) ($p \in [0, 1]$), extending the call-by-value evaluation of (regular) lambda calculus.

An *applicative context* is a "term with a hole" pointing where evaluation can happen. For instance, in our setting we do not allow rewriting under lambdas. Applicative contexts are defined according to the following grammar.

$$\begin{array}{rcl} C[-] & ::= & [-] \mid MC[-] \mid C[-]V \mid \langle C[-], N\rangle \mid \langle V, C[-]\rangle \mid \\ & & \mathtt{let}\ \langle x, y\rangle = C[-]\ \mathtt{in}\ N \mid \mathtt{let}\ \langle\rangle = C[-]\ \mathtt{in}\ N \mid \\ & & \mathtt{if}\ C[-]\ \mathtt{then}\ N\ \mathtt{else}\ P. \end{array}$$

The rewrite system consists of two parts: the "classical" part, not interacting with Q and L, and the "quantum" part, whose goal is to emulate the interaction with the quantum coprocessor. We define a first rewrite system $\to_c$ on terms characterizing the classical part of the evaluation, as follows.

$$\begin{array}{lcl} C[(\lambda x.M)V] & \to_c & C[M[x := V]] \\ C[\mathtt{let}\ \langle\rangle = \langle\rangle\ \mathtt{in}\ M] & \to_c & C[M] \\ C[\mathtt{let}\ \langle x, y\rangle = \langle V, W\rangle\ \mathtt{in}\ M] & \to_c & C[M[x := V, y := W]] \\ C[\mathtt{if}\ \mathtt{tt}\ \mathtt{then}\ M\ \mathtt{else}\ N] & \to_c & C[M] \\ C[\mathtt{if}\ \mathtt{ff}\ \mathtt{then}\ M\ \mathtt{else}\ N] & \to_c & C[N] \\ C[\mathtt{let\ rec}\ f\ x = M\ \mathtt{in}\ N] & \to_c & C[N[f := \lambda x.\mathtt{let\ rec}\ f\ x = M\ \mathtt{in}\ M]] \end{array}$$

We can then define the rewrite system on programs as first

$$M \to_c N \qquad \text{implies} \qquad [Q, L, M] \to_1 [Q, L, M']$$

for the classical part, and for the quantum part, assuming $Q \in \mathcal{Q}^{\otimes n}$ and z is fresh:

$$\begin{aligned}
[Q, L, C[\mathtt{qinit}\,\mathtt{tt}]] &\to_1 [Q \otimes |1\rangle, L \cup \{z \mapsto n+1\}, C[z]], \\
[Q, L, C[\mathtt{qinit}\,\mathtt{ff}]] &\to_1 [Q \otimes |0\rangle, L \cup \{z \mapsto n+1\}, C[z]], \\
[Q, L, C[U\,x]] &\to_1 [(U \otimes I)Q, L, C[x]] && \text{if } U \text{ is unary and } L(x) = 1, \\
[Q, L, C[U\,\langle x, y\rangle]] &\to_1 [(U \otimes I)Q, L, C[\langle x, y\rangle]] && \text{if } U \text{ is binary, } L(x) = 1, L(y) = 2, \\
[Q, L, C[\mathtt{meas}\,x]] &\to_{|\alpha_b|^n} [Q_b, L, C[b]],
\end{aligned}$$

when $Q = \alpha_{\mathtt{ff}} |0\rangle \otimes Q_{\mathtt{ff}} + \alpha_{\mathtt{tt}} |1\rangle \otimes Q_{\mathtt{tt}}$ with $Q_{\mathtt{ff}}$ and $Q_{\mathtt{tt}}$ normalized.

The language satisfies the usual safety properties: subject reduction and progress.

Example D.7. The type system is designed to not allow the duplication of quantum bits. The type $!\mathtt{qbit}$ is therefore empty: there is no closed term M such that $\vdash M : !\mathtt{qbit}$. This property heavily relies on the constraint we placed on the promotion rule (p): one can only duplicate *values*. It is however possible to build a duplicable term of type $!(\mathbf{1} \multimap \mathtt{qbit})$, as for instance

$$\vdash \lambda x.(\mathtt{let}\ \langle\rangle = x\ \mathtt{in}\ H(\mathtt{qinit}\,\mathtt{ff})) : !(\mathbf{1} \multimap \mathtt{qbit})$$

is derivable. We come back to this example in Example D.9

D.1.3 Cut-elimination and Curry-Howard Isomorphism

Besides the cut-rule (and (ex_σ), which does not count), the proof rules of linear logic are structural: they construct a sequence of formulas out of more primitive ones. One important question in logic is, given a proof π, whether one can rewrite it to obtain a *cut-free* proof, using only structural rules (and exchange rules). This problem is known as *cut-elimination*. In the case of LL (and LL+mix), one can equip the set of proofs with a strongly normalizing and confluent rewriting system whose normal forms are precisely cut-free proofs [GLT90]. For the sake of the presentation, we only discuss two of them: the interaction between (cut) and (ax)

$$\dfrac{\dfrac{}{\vdash A, A^\perp}\,(\mathrm{ax}) \qquad \begin{matrix}\pi \\ \vdots \\ \vdash A\end{matrix}}{\vdash A}\,(\mathrm{cut}) \qquad \to \qquad \begin{matrix}\pi \\ \vdots \\ \vdash A\end{matrix}$$

and the rewriting of a cut between $(\otimes)$ and $(\parr)$

$$\dfrac{\dfrac{\begin{matrix}\pi_1 \\ \vdots \\ \vdash \Delta_1, A\end{matrix} \qquad \begin{matrix}\pi_2 \\ \vdots \\ \vdash \Delta_2, B\end{matrix}}{\vdash \Delta_1, \Delta_2, A \otimes B}\,(\otimes) \qquad \dfrac{\begin{matrix}\pi_3 \\ \vdots \\ \vdash \Gamma, A^\perp, B^\perp\end{matrix}}{\vdash \Gamma, A^\perp \parr B^\perp}\,(\parr)}{\vdash \Delta_1, \Delta_2, \Gamma}\,(\mathrm{cut}) \qquad \to \qquad \dfrac{\begin{matrix}\pi_1 \\ \vdots \\ \vdash \Delta_1, A\end{matrix} \qquad \dfrac{\begin{matrix}\pi_2 \\ \vdots \\ \vdash \Delta_2, B\end{matrix} \qquad \begin{matrix}\pi_3 \\ \vdots \\ \vdash \Gamma, A^\perp, B^\perp\end{matrix}}{\vdash \Gamma, \Delta_2, A^\perp}\,(\mathrm{cut})}{\vdash \Delta_1, \Delta_2, \Gamma}\,(\mathrm{cut}) \tag{D.9}$$

Remembering that the linear implication $A \multimap B$ is built as $A^\perp \parr B$, note how the rewrite rule shown in Eq. (D.9) corresponds to a form of $\multimap$-elimination. This in fact precisely corresponds to the beta-rule of the lambda calculus, such as in the presentation of Section B.3.1 (Although the relation is slightly non-trivial [Reg92] and linked to explicit substitution [CK97, Acc15])

This *Curry-Howard correspondence* for linear logic has been analyzed by many authors [Abr93, BBHP92, BBPH93, Bie93, Wad93]. Formalizing the intuition drawn in Example D.1,

type systems based on linear logic make it possible to specify whether resources are used only once: a function of type $A \multimap B$ is guaranteed to use its argument exactly once, while the type $\alpha \multimap (\alpha \otimes \alpha)$ is empty. Many refinements or extensions are possible. For instance, one can relax the linearity constraint and allow weakening —i.e. erasure— of linear resources to get *affine linear logic* [Tro92]. One can use the exponential modality to characterize implicit computational complexity [AR02, BT04, Gir98, Laf04, LMZ10], or add annotations to exponentials to keep track of the number of uses of a particular resource with bounded linear logic [GSS92, LH10] or discuss differential privacy [RP10, GHHNP13]. And, as exemplified in [PSV14] and the sketch of Section B.3, one can distinguish between duplicable and non-duplicable data and apply it to quantum computation and the manipulation of qubits.

D.2 A Denotational Semantics

This section is devoted to the study of a *denotational* semantics for the quantum lambda calculus. A denotational semantics is an interpretation of programs as mathematical functions, composition of programs corresponding to function composition. Denotational semantics are expressive tools to bridge programming languages with logical theories through the Curry-Howard correspondence. By exhibiting the compositional structures underlying a language, a denotational semantics validates the soundness of its design.

One of the challenge in semantics is the *compatibility* of quantum and classical features when intertwined, as exemplified in the quantum lambda calculus. On the one hand, the typical semantics for quantum computation relies on linear maps and positive operators in finite dimension. On the other hand, classical information should be duplicable, therefore requiring some notion of non-linearity. Finally, the mix of quantum information within classical datatypes such as lists entails non-standard objects such as infinite datatypes of list of qubits, hinting at the need for infinite dimensional vector spaces.

In this section, we present our solution for such a denotational semantics. We follow an iterative approach, starting with a quick review of the previous existing approaches (Section D.2.1). We then present our work: a simple semantics based on completely positive maps (Section D.2.2) to which we progressively add constructs: additives (Section D.2.3), recursive datatypes (Section D.2.4), and duplication (Section D.2.5). We conclude with a discussion on other possible approaches (Section D.2.6).

> This section is the result of a long gestation. It started at the end of my Ph.D thesis with the design of a CPM-based semantics for a purely linear quantum lambda calculus [SV08a]. The adjunction of recursive datatypes and duplication was then a roadblock for a long time: how to include them in a sound way within a finite-dimensional setting? The knot was untangled in 2014 with the development of general techniques for quantitative semantics of linear logic and semantics of probabilistic PCF [ETP14]. With Michele Pagani, we were able to port these technique to the quantum case and answer the problem [PSV14].

D.2.1 Background on Denotational Semantics

Modeling higher-order languages have historically been done using *domains* and continuous lattices. Algebraic effects, such as probabilities, can be handled with the use of a suitable monad [Jon90, GHKLMS03] (although this requires some care [Gra88, JT98]). On the other hand, linear algebra and functional analysis have been from the very beginning an extensional target model for linear logic. Originally designed for System F [Gir86], coherent spaces —at the root of the design of linear logic [Gir87]— have soon been generalized to support algebraic effects [Gir99, Ehr02, Gir04, DE11, EPT11]. The other original semantics for linear

logic, quantitative domains [Gir88], has also spurred many rich algebraic models: Fock spaces [BPS93a], Hopf algebras [Blu96], Köthe spaces [Ehr02], finiteness spaces [Ehr05], *etc.* It makes it possible to prove fine-grained quantitative properties of programs [LMMP13].

From a categorical perspective, building a model of linear logic requires one to accommodate four components: the multiplicative fragment, the additive fragment, the modalities and the involution —the last one being optional if the target is *intuitionistic* linear logic. Depending on the computational objective, one can also ask for traces, and/or fixpoints, and/or allow affine behavior, *etc.*

As the type system of quantum lambda calculus is based on linear logic, it is reasonable to look for a suitable algebraic model of linear logic capturing quantum effects. One of Girard's goals is to bridge physics and logic: Instead of trying to organically extract logical structure out of positive operators and quantum observables [BN36, Mit78, DG02] —a problematic approach from a computational perspective [Abr07]— Girard started from the desired logical structures and built a semantics inspired from quantum structures: *quantum coherent spaces* [Gir04]. Although such a quantum-based semantics is expressive enough to model (restricted forms of) modalities [Bar10], Selinger [Sel04b] showed that it is not adequate for modeling quantum computation, as it is missing some entangled states —for instance $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

D.2.2 CPM as Compact Closed Category

As discussed in Section B.1.3, linear distributions of pure states are adequately represented with trace-1 positive matrices. Selinger discusses how possibly non-terminating quantum programs can then be modeled with trace-non-increasing *completely positive maps*: a *superoperator*. A completely positive map (CPM) $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{m\times m}$ is a linear map such that for all k, the map $\mathrm{id}_{\mathbb{C}^{k\times k}} \otimes f$ sends positive matrices to positive matrices. Let us discuss a few aspects of this definition.

1. A superoperator might therefore output a positive matrix of trace strictly less than one: this trace corresponds to the overall probability of termination of the corresponding algorithm.

2. Consider a valid quantum algorithm P of input A and of output B. One can construct another valid quantum algorithm with a dummy variable C: the resulting algorithm P' inputs in $C \otimes A$ and outputs in $C \otimes B$. The denotation $[\![P']\!]$ is equal to $\mathrm{id}_{[\![C]\!]} \otimes [\![P]\!]$. This is the reason for the second constraint on superoperator.

The trace-non-increasing constraint gives a fully complete model for first-order quantum computation, as discussed in [Sel04a]. In the case of higher-order quantum computation, we drop this constraint and work instead with general, completely positive maps. Indeed, in order to model functions we can then rely on the Choi theorem [Cho75], stating that

Theorem D.8 ([Cho75, Th. 2]). *Let f be a linear map from $\mathbb{C}^{n\times n}$ to $\mathbb{C}^{m\times m}$. Then f is completely positive if and only if $\chi_f \in \mathbb{C}^{mn\times mn}$ blockwise defined as*

$$\chi_f = \begin{pmatrix} fE_{1,1} & \cdots & fE_{1,n} \\ \vdots & \ddots & \vdots \\ fE_{n,1} & \cdots & fE_{n,n} \end{pmatrix}$$

is positive, where $E_{i,j} \in \mathbb{C}^{n\times n}$ is the matrix with 0s everywhere apart for one 1 on the i-th line and j-th column. □

Using Theorem D.8, one can design a model of MLL using positive matrices and completely positive maps, as follow. We define the category $\mathbf{CPM}$ with the following data:

- Objects: natural numbers;

- Morphisms: $f : n \to m$ is a completely positive map $\mathbb{C}^{n\times n} \to \mathbb{C}^{m\times m}$.

CPM can be equipped with a monoidal structure, behaving as the (usual) multiplication on integers and as Kronecker product on morphisms. Thanks to Theorem D.8, the functor $A \otimes (-)$ admits a right adjoint, according to the natural isomorphism

$$\mathbf{CPM}(A \otimes B, C) \simeq \mathbf{CPM}(A, B \otimes C).$$

This makes **CPM** *compact closed* [Sel07], model of MLL. Note however that the model is degenerated as $\parr$ and $\otimes$ coincide.

D.2.3 Accommodating the Additives

In order to be able to at least manipulate Boolean values, we need to extend **CPM** to accommodate the additives. Since we are in the context of finite dimensional vector spaces, the easiest is to consider the *biproduct completion* $\mathbf{CPM}^{\oplus}_{\text{fin}}$ of **CPM**:

- Objects: lists of natural numbers $\sigma = n_1, \dots, n_k$
- Morphisms: If $\sigma = n_1, \dots, n_k$ and $\tau = m_1, \dots, m_{k'}$, then $f : \sigma \to \tau$ is a family $f = \{f_{i,j}\}_{i,j}$ where $f_{i,j} : n_i \to m_j$ is a **CPM**-morphism.
- Composition is obtained with matrix multiplication:

$$\{f_{i,j}\}_{i,j} \circ \{g_{j,k}\}_{j,k} = \left\{ \sum_j f_{i,j} \circ g_{j,k} \right\}_{i,k} \tag{D.10}$$

  whereas the identity is a diagonal matrix of identities.

The compact-closed structure of **CPM** carries over to $\mathbf{CPM}^{\oplus}_{\text{fin}}$ in a straightforward manner:

$$(n_1, \dots, n_k) \otimes (m_1, \dots, m_l) = n_1 m_1, \dots, n_1 m_l, n_2 m_1, \dots, n_2 m_l, \dots\dots, n_k m_l$$

The category $\mathbf{CPM}^{\oplus}_{\text{fin}}$ makes a model of MALL, albeit degenerate since both the multiplicatives and the additives collapse. However, it makes a fully-abstract model of a *strictly linear* lambda calculus, as shown in [SV08a].

D.2.4 Accommodating Recursive Datatypes

If the category $\mathbf{CPM}^{\oplus}_{\text{fin}}$ can accommodate additives, the system is restricted to *finite* biproducts $\bigoplus_{i=1}^{n} A_i$. This limits the expressiveness of the system: recursive datatypes such as lists:

$$[A] = \bigoplus_{n=0}^{\infty} A^{\otimes n} \tag{D.11}$$

cannot be represented as they require *infinite* biproducts. Following the same intuition as for the construction of $\mathbf{CPM}^{\oplus}_{\text{fin}}$, an infinite biproduct would correspond to having *infinite* lists of natural numbers for objects, and infinite-dimensional matrices. The difficulty then comes with the composition, as we now end up with an infinite sum in Eq. (D.10). Indeed, in general, the sum might not converge: consider for instance $f : 1, 1, 1 \dots \to 1, 1, 1 \dots$ defined as $f_{i,j} : 1 \to 1$ being the identity for all i, j. The composition of f with itself does not converge.

The solution we propose in [PSV14] consists in first completing **CPM** with "all possible infinite" elements. Recall that positive matrices admit a natural ordering: Löwner order. This can be pointwise ported to completely positive maps: each homset $\mathbf{CPM}(A, B)$ is a Löwner positive cone. One interesting property of the Löwner order is that such a positive cone is *bounded*

directed complete: (1) there is a minimum element (the 0 function), and (2) any bounded directed subset $D \in \mathbf{CPM}(A, B)$ admits a least upper bound.

Formally, the completion we consider is the *D-completion* [ZF10]. For the purpose of the discussion, we are interested in two properties: First, the D-completion is functorial, and then it preserves existing least upper bounds. On other words, the only additional elements are "at infinity" —precisely what we need.

We can therefore define the category $\mathbf{CPM}_D$ as follows: the objects are those of $\mathbf{CPM}$, and the morphisms from n and m are exactly the elements of the D-completion of $\mathbf{CPM}(n, m)$. The homsets $\mathbf{CPM}_D(n, m)$ are now dcpos: generalized sums are always defined. This then makes it possible to define the *infinite biproduct completion* $\mathbf{CPM}_D^{\oplus}$ of $\mathbf{CPM}_D$ exactly as desired: objects are infinite lists of objects of $\mathbf{CPM}_D$, and morphisms are infinite-dimensional matrices. The composition is defined as in Eq. (D.10), and the possibly infinite sum resulting from the definition is well-defined.

D.2.5 Accommodating Duplication

With infinite coproducts we can encode the behavior type $!A$ inside the type of lists shown in Eq. (D.11). Indeed, a duplicable element of type $!A$ can be regarded as the biproduct of zero copies of the element, one copy of the element, two copies of the element, *etc.*

The typical example is the program that inputs a coin, tosses it twice and computes the conjunction of the results. In $\mathbf{CPM}$ the type of a coin is $1, 1$: a pair of two probabilities (a, b), where a is the probability of getting $\mathtt{tt}$ and b the probability of getting $\mathtt{ff}$. The aforementioned program therefore corresponds to the (non-linear) map

$$(a, b) \longmapsto (a^2, 2ab + b^2).$$

The reason for the non-linearity is the identification of `bit` and `!bit`. Instead, we can consider a more expressive representation for the input, as

$$(a_*, a_{\mathtt{tt}}, a_{\mathtt{ff}}, a_{\mathtt{tt},\mathtt{tt}}, a_{\mathtt{tt},\mathtt{ff}}, a_{\mathtt{ff},\mathtt{tt}}, a_{\mathtt{ff},\mathtt{ff}}, a_{\mathtt{tt},\mathtt{tt},\mathtt{tt}}, a_{\mathtt{tt},\mathtt{tt},\mathtt{ff}}, a_{\mathtt{tt},\mathtt{ff},\mathtt{tt}} \ldots) \in \mathbf{1} \oplus \mathtt{bit} \oplus (\mathtt{bit} \otimes \mathtt{bit}) \oplus \cdots.$$

A duplicable coin producing $\mathtt{tt}$ with probability a and $\mathtt{ff}$ with probability b is now represented as the sequence

$$(a + b, a, b, a^2, ab, ab, b^2, \ldots)$$

and the aforementioned program has now for semantics

$$(a_*, a_{\mathtt{tt}}, a_{\mathtt{ff}}, a_{\mathtt{tt},\mathtt{tt}}, a_{\mathtt{tt},\mathtt{ff}}, a_{\mathtt{ff},\mathtt{tt}}, a_{\mathtt{ff},\mathtt{ff}}, \ldots) \mapsto (a_{\mathtt{tt},\mathtt{tt}}, a_{\mathtt{tt},\mathtt{ff}} + a_{\mathtt{ff},\mathtt{tt}} + a_{\mathtt{ff},\mathtt{ff}}),$$

now a linear, completely positive map.

Such a construction is however failing in providing the required categorical structure of comonoid. Indeed, if $!A$ is modeled with $[A]$, two copies of A can very well be distinct.

Instead of a plain tensor, what is needed for $!A$ is a *symmetric tensor* [MTT09], connected to Fock spaces and used e.g. for modeling probabilistic programs in probabilistic coherent spaces [ETP14]. Considering the case of `!bit`, it corresponds to define

$$!\mathtt{bit} \triangleq \mathbf{1} \oplus \mathtt{bit} \oplus \mathtt{bit}^{\odot 2} \oplus \mathtt{bit}^{\odot 3} \oplus \cdots,$$

where $\mathtt{bit}^{\odot n}$ is the equalizer of

$$\mathtt{bit}^{\odot n} \longrightarrow \mathtt{bit}^{\otimes n} \underset{\text{symmetry}}{\overset{\text{symmetry}}{\rightrightarrows}} \mathtt{bit}^{\otimes n}$$

For instance, $\mathtt{bit}^{\odot 2}$ corresponds to the subcone of $1, 1, 1, 1$ invariant under swap: this corresponds to

$$\{\, (a, b, b, c) \;\big|\; a, b, c \in [0, 1] \text{ and } a + 2b + c \leq 1 \,\}$$

or, equivalently,

$$\{\, (a, b) \otimes (a, b) \;\big|\; a, b \in [0, 1] \text{ and } a + b \leq 1 \,\}.$$

In our case, this requires to modify the original category $\mathbf{CPM}$ to account for such equivalence classes: we invite the reader to consult the paper for more information [PSV14]. In any case, the resulting model is shown to be adequate for a quantum lambda calculus of the form presented in Section D.1.2 together with coproducts and recursive types. The model is in fact richer than what we originally showed: it is fully-abstract [CV20].

Example D.9. In Example D.7 we discuss the type $!(1 \multimap A)$. On the semantic side, in $\mathbf{CPM}_D$, this type is literally equal $!A$. We can reconciliate this fact with the non-duplicability of qubits by realizing that the semantics $\mathbf{CPM}_D$ is richer than what can be expressed in the quantum lambda-calculus. In particular, the semantics can support not only call-by-value but also call-by-name. The semantics of $!\mathtt{qbit}$ then represents *thunks* of terms computing a qubit. On the language side, in a call-by-value perspective such a term has to be encapsulated inside a lambda-abstraction: we fall back on $!(1 \multimap \mathtt{qbit})$.

D.2.6 Discussion

To overcome the finite-dimensional limitation of $\mathbf{CPM}$, we used in [PSV14] abstract constructions based on category and domain theory. If one can argue that we only added "infinite" elements that are anyway not representable by programs, it can be regarded as a limitation of our approach. Clairambault and de Visme [CV20] offer an alternative approach based on event structures [Win80, Win87] and game semantics that does not require infinite dimensional spaces. It is worth noting that their approach solves a long-standing issue in quantum game semantics: capturing entangled elements within the tensor [DP08, Del11, Del08b].

Other approaches rely on generalizations of $\mathbf{CPM}$: C^* and von Neumann algebras. Westerbaan *et al.* [Wes16] discusses how to recover the required structures, while [Wes19, Sec. 4.3] describe a model for the quantum lambda calculus in this framework. Finally, lately Pechoux *et al.* [PPRZ20] discusses how to build recursive types with von Neumann algebras. One can also mention the presheaf model of Malherbe [Mal10, MSS13] and the categorical construction of Hasuo and Hoshino based on Geometry of Interaction [HH11, HH17].

D.3 An Executable Semantics

This section is devoted to the description of a low-level, operational semantics for a quantum lambda calculus. As discussed in Section D.1.2, the standard computational interpretation of the quantum lambda calculus is a rewrite system based on variable substitution. We discuss here how to retrieve a circuit-based interpretation of a quantum lambda-term, using a technique stemming from the study of linear logic: the *Geometry of Interaction* (GoI). Originating from Girard [Gir89], GoI has shown useful in understanding the relationship between high-level constructs and low-level, assembly-like presentations [Mac95, GSS11].

Section D.3.1 first presents the graphical notation of proof-nets for representing proofs of linear logic. Section D.3.2 discusses how typed lambda-terms can be interpreted as proof-nets. Section D.3.3 describes the token-machine presentation of GoI [DR99], giving a graph-based, executable semantics for programs when translated to proof-nets. Section D.3.4 exposes the limit of the standard approach, as it does not support token synchronization: this is required for quantum computation. Section D.3.5 offers a generic solution, and Section D.3.6 discusses the solution we build specifically for quantum computation.

The strength of our proposal is to unfold the circuit-like structure hidden inside quantum lambda-terms: tokens follow the tangled wires of the circuit.

D.3.1 Proof-Nets for MELL

One of the nagging issues with proofs of linear logic is the exchange rules: as discussed in Remark D.2 it is essential, yet it looks like an unnecessary, bureaucratic construct. An alternative, graphical presentation of proofs of linear logic consists in *proof nets*. For more information on proof-nets, consult e.g. Laurent's notes [Lau13], from which this section takes inspiration.

A proof net is a *proof structure* with a *validity criterion*. A proof structure is a directed graph, possibly piecewise connected, with labeled edges and nodes. Thanks to the graph structure, there is no need for permuting anything, and the possible ambiguity of Eq (D.1) disappears. In the context of this thesis, we will concentrate on the multiplicative exponential fragment of linear logic (MELL), without units. A proof structure for MELL is built out of the nodes of Figure D.1.

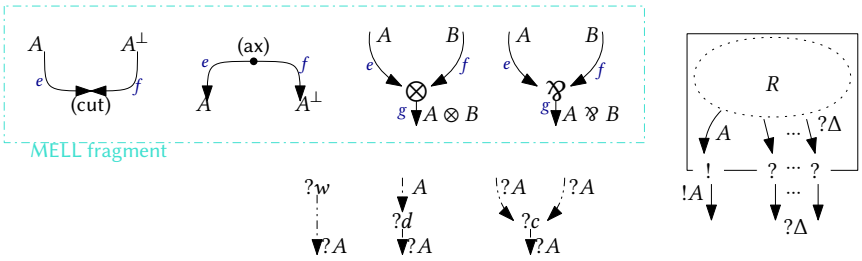


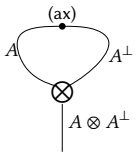
Figure D.1: Constructors for MELL proof structures.

When clear, we shall omit the arrows symbol. Input edges are called *premises* and output edges *conclusions*; the edges of a node are ordered. The nodes (cut) and (ax) corresponds to the similarly named proof rules, while the nodes $\otimes$ and $\parr$ decompose multiplicative formulae. The edges for the MLL fragment (in the Turquoise dashed box) are named e, f and g: they shall be used in Section D.3.3. The four other nodes are for managing modalities. The right-most node, the *box-node*, stands for the promotion proof rule (p). The box encapsulates a proof-structure R. The wires going in the box goes through *doors*: left-most one is the *principal* door while the other are *auxiliary* doors. The remaining nodes $?w$, $?d$ and $?c$ respectively stand for weakening, dereliction and promotion. We extend the notion of premises and conclusions to proof-structures: if the premises of the structure S are $A_1, \dots A_n$ and the conclusions are $B_1, \dots, B_m$, we say that S corresponds to the sequent $\vdash A_1^\perp, \dots A_n^\perp, B_1, \dots, B_m$.

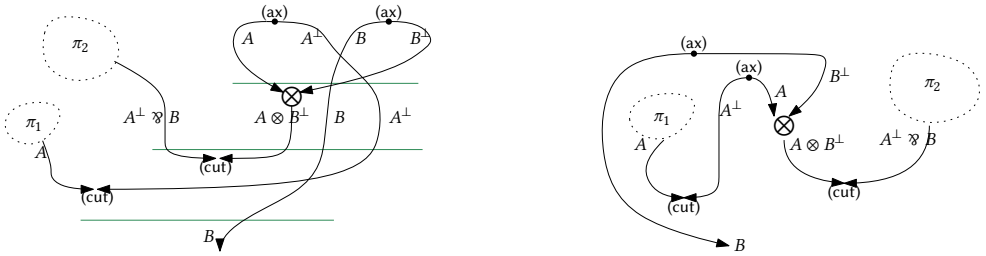
Note how each node corresponds to a proof rule (apart for the exchange rule). A proof can then be directly transposed into a proof structure. For instance, Fig. D.2 the proof of Eq. (D.2) becomes the proof net shown in Figure D.2b. As it is a graph, it is the same as the net in Figure D.2a.

Remark D.10. In the case where we have constant formulas α in the grammar of the logic, we can add specific nodes to reflect the corresponding proofs.

The fact that a sequent $\vdash \Delta$ admits a proof structure does not however always imply that there exists a proof for it: consider for instance the proof structure



(D.12)



(a) Following the proof (b) Untangled

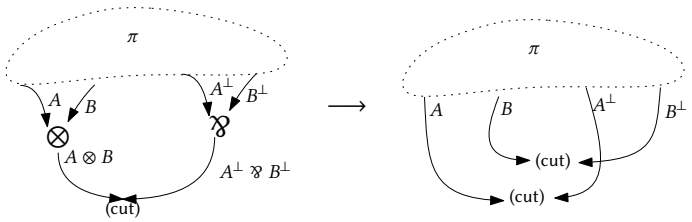
Figure D.2: Proof-net corresponding to Eq. (D.2)

associated to the invalid sequent $\vdash A \otimes A^\perp$.

Characterizing *proof nets*, i.e. proof structures representing a valid proof for a given sequent, requires a *validity criterion*: many proposals [NM07] have been proposed since the original Girard's *longtrip condition* [Gir87]. Originally developed for MLL [DR89] but generalizable to MELL [Dan90, GM01], a versatile criterion is Danos&Regnier's *switching condition*. It uses the notion of *path*: a path in a proof structure π is a sequence of nodes, pairwise connected with edges. In the case of MLL, it is called *switching* if its does not go through both premises of a $\parr$-node. A proof-structure π is called *switching acyclic* when it does not contain switching cyclic paths. In this case, we call it a *proof net*.

Example D.11. The proof structure presented in Eq. (D.12) is not switching acyclic: it does not correspond to a proof of $\vdash A \otimes A^\perp$.

In Section D.1.3, we discussed cut-elimination for the proofs of sequents: a similar procedure can be designed for proof-nets. For instance, the rewrite rule shown in Eq. (D.9) becomes for proof-nets



(D.13)

The validity criterion are preserved through the reduction: a valid proof-structure remains valid through rewriting.

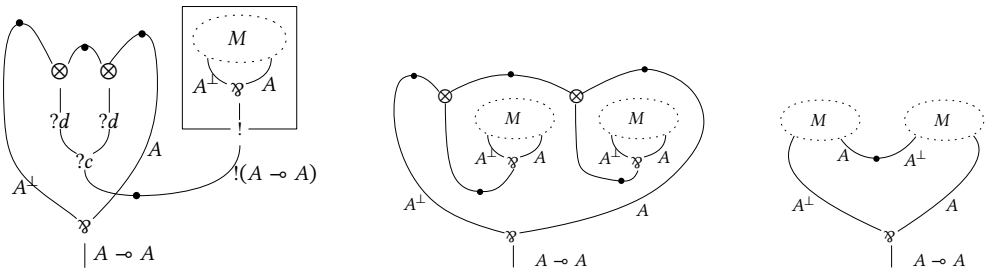
There have been plenty of works and extensions of proof-nets: interaction nets [Laf90, Laf95, Maz06], differential nets [ER06, Tra11], *etc.* It is a flexible structure able to capture many logical aspects while leaving out much of the bureaucracy of proofs.

D.3.2 Encoding Higher-Order Languages

Being a graphical representation, proof-nets make an easily extensible, versatile representation for programs. Typically, as mentioned in Remark D.10 one can add to the graphical language nodes representing constants and opaque operators, and update the rewriting system accordingly. Similarly, one can extend the boxing constructs to other situations, such as dealing with tests [LFHY14] and recursion [LFVY15].

A proof-net directly comes from the typing derivation: in the case of the quantum lambda calculus, as the type system is based on linear logic the translation is immediate. For instance, the term

$$\vdash (\lambda f.\lambda x.f(fx))\lambda w.M : A \multimap A$$



(a) Original (b) Opening boxes (c) Unfolding λ's

Figure D.3: Translation of $\vdash (\lambda f.\lambda x.f(fx))\lambda w.M : A \multimap A$

corresponds to the proof-net shown in Figure D.3. Figure D.3a corresponds to the original term (modulo some yanking for legibility). The duplicated subterm $\lambda w.M$ is the box on the right of the cut, while the contraction on the left corresponds to the duplication of the variable f. The derelictions "open" the two copies of the box. The result of the copy and the opening of the boxes is shown in Figure D.3b: it corresponds to $\lambda x.(\lambda w.M)((\lambda w.M)x)$. Finally, Figure D.3c shows the beginning of the unfolding of the two "$\lambda w.$".

D.3.3 Token-based Geometry of Interaction

Geometry of Interaction (GoI) stands as one of Girard's flagship research projects. Its main goal consists in extracting the *computational content* of a proof, stable under cut-elimination. If Girard attacked this problem under many different angles [Gir89, Gir90, Gir95a, Gir03, Gir11], the one we consider in this chapter is the *token-based GoI* [Gir89]. In this approach, GoI can be seen as a procedure to construct a sequential data-flow machine, with a token running in a proof net. In particular, it draws a direct link between high-level programming constructs and low-level, assembly languages [DR99, Mac95, Mac94] —it has even been used as a backbone for designing compilers, with support for higher-order functions [GS11], local, assignable states [Ghi07], concurrency [GS10], recursion [GSS11], *etc.*

In order to illustrate the difference between the standard approach and the contribution presented in Section D.3.5, we propose a brief introduction of the *IAM* [DR99] —the *Interaction Abstract Machine*—. We focus for this presentation on the multiplicative fragment MLL of MELL —that is, without modalities. In this section, to relate with Section D.1.2, we follow the notation presented in [LFVY15].

In the MLL fragment we consider, a formula is given by the grammar

$$A, B \quad ::= \quad \alpha \mid \alpha^{\perp} \mid A \otimes B \mid A \parr B.$$

For the purpose of the discussion, and in line with Section D.1.2 we replace the units with constants α, ranging over fixed set of identifiers. A proof-net consists of the nodes (cut), (ax), ($\otimes$), ($\parr$), together with a dummy node (α) with one conclusion of type α and no premises.

The state of an IAM on a proof-net π consists of a triple (e, s, d) where e is an edge of π, s an address and d is a direction $\uparrow$ or $\downarrow$. An address is a stack: a list of the literals l and r. Cons is denoted with ":", while the empty stack is ε. An address represents the position of a subformula inside a formula. For instance, the address $r : l : \varepsilon$ points to α_3 in the formula $(\alpha_1 \otimes \alpha_2) \parr (\alpha_3 \otimes \alpha_4)$.

A (reversible) rewrite system for IAM states is then derived from the structure of a net. Following the naming convention for edges shown in Figure D.1, the rules for the token movements in MLL proof nets are shown in Table D.4. An initial (resp. final) state of the IAM on an MLL-net π consists in a state of the form $(e, s, \uparrow)$ (resp. $(e, s, \downarrow)$), where e is a conclusion edge

(cut)	$(e,s,\downarrow) \to (f,s,\uparrow)$	$(f,s,\downarrow) \to (e,s,\uparrow)$
(ax)	$(e,s,\uparrow) \to (f,s,\downarrow)$	$(f,s,\uparrow) \to (e,s,\downarrow)$
up ($\otimes$)	$(g,l{:}s,\uparrow) \to (e,s,\uparrow)$	$(g,r{:}s,\uparrow) \to (f,s,\uparrow)$
up ($\parr$)	$(g,l{:}s,\uparrow) \to (e,s,\uparrow)$	$(g,r{:}s,\uparrow) \to (f,s,\uparrow)$
down ($\otimes$)	$(e,s,\downarrow) \to (g,l{:}s,\downarrow)$	$(f,s,\downarrow) \to (g,r{:}s,\uparrow)$
down ($\parr$)	$(e,s,\downarrow) \to (g,l{:}s,\downarrow)$	$(f,s,\downarrow) \to (g,r{:}s,\uparrow)$

Table D.4: Rules for the IAM token Machine, MLL fragment.

of π and s points to a constant subformula of the formula attached to e. We write $\mathcal{I}$ the set of initial states and O the state of final states.

Proposition D.12. *If π is an MLL-proof-net, the IAM machine deterministically sends initial states to final states: it induces a bijection $\Sigma_\pi : \mathcal{I} \to O$. Furthermore, this bijection is invariant under cut-elimination.* □

Figure D.5 shows the behavior of the IAM machine on the proof-net corresponding to the cut of the proof of $\alpha_1 \otimes \alpha_2 \vdash \alpha_2 \otimes \alpha_1$ and the proof of $\alpha_2 \otimes \alpha_1 \vdash \alpha_1 \otimes \alpha_2$. This gives the identity on $\alpha_1 \otimes \alpha_2$. The token machine "realizes" the computation. Note for instance how the initial state $(e, r{:}\varepsilon, \uparrow)$ sitting on α_2 goes to the terminal state $(o, r{:}\varepsilon, \downarrow)$, corresponding to a token also sitting on α_2. Note also how this is invariant under the rewrite rule shown in Eq. (D.13).

This example generalizes: the IAM rewrite system on a proof-net π *realizes* the computation described by the corresponding proof. The abstract machines stemming from such an approach follow a *call-by-name* strategy [DR99, Mac95]: arguments are passed to their calling functions without being evaluated. Formally, [DR99] makes a connection between the IAM and the *Krivine abstract machine* (KAM) [Kri07].

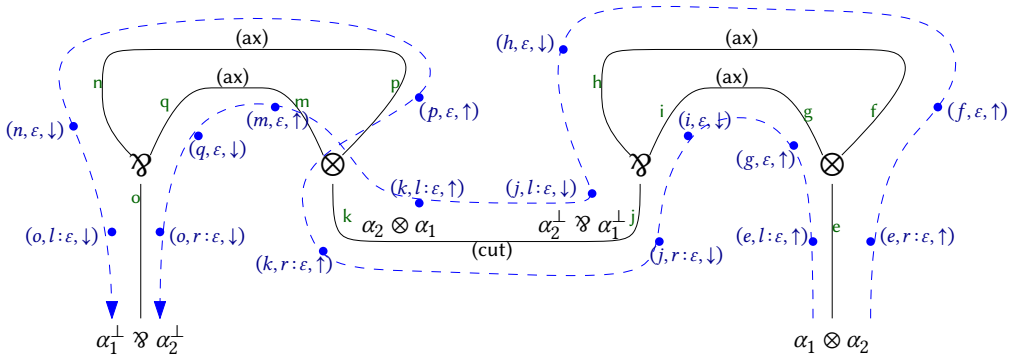


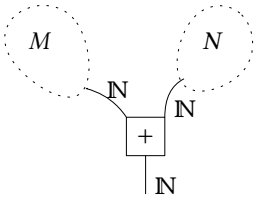
Figure D.5: A run of the IAM token machine.

D.3.4 Limits of the Conventional Approach

For our purpose, the two main limits to this conventional, stateless token-based presentation of GoI are the strict sequentiality of the machinery and the fact that it is call-by-name. As in the case of game semantics [AM97], directly handling call-by-value —without specific encoding, such as CPS [Wad03]— typically requires side-effects [Sch14, HMH14].

The strict sequentiality is a problem in the context of additional nodes reflecting operations on atomic types. Suppose for instance that one of the type α stands for the natural

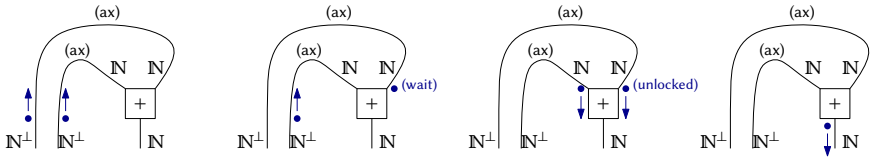
numbers $\mathbb{N}$: we can add to token states a register holding a natural number, and have a special node "+" for addition. If M and N are terms of type $\mathbb{N}$, the term $\vdash M + N : \mathbb{N}$ then corresponds to some net



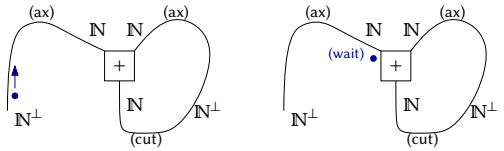
Computing with an IAM-style machine requires to start from the conclusion; but what should we do when we reach the +-node? Should we go left, right? Traditional solutions involve making an arbitrary choice on which premise the token should explore first. However, despite the fact that this requires a state to store the intermediate result, in languages such as the quantum lambda calculus, this is not always possible as some operators act on non-atomic types —for instance, 2-qubit unitary gates act on $\mathtt{qbit}\otimes\mathtt{qbit}$. This makes it difficult to adapt to the single-token IAM, and requires a novel approach.

D.3.5 Multi-Token Geometry of Interaction

In order to answer the sequentiality problem listed in Section D.3.4, dal Lago *et al.* [LFHY14] offers an alternative approach for a Geometry of Interaction: Instead of starting from conclusion to fetch values, values "flow" on their own from inputs *towards* conclusions —in a call-by-value spirit. Instead of one single token, the GoI machine of [LFHY14] fires one token per potential value. Tokens are then emitted from negative conclusions and, if any, from nodes introducing atomic types. This solves the problem discussed in Section D.3.4: each one of the premises of the +-node eventually meets with a value-token. The problem [LFHY14] addresses —in the very restricted case of MLL— is the synchronization issue: for the +-node to fire, it needs *both* of its argument-tokens to have arrived, as shown in this example

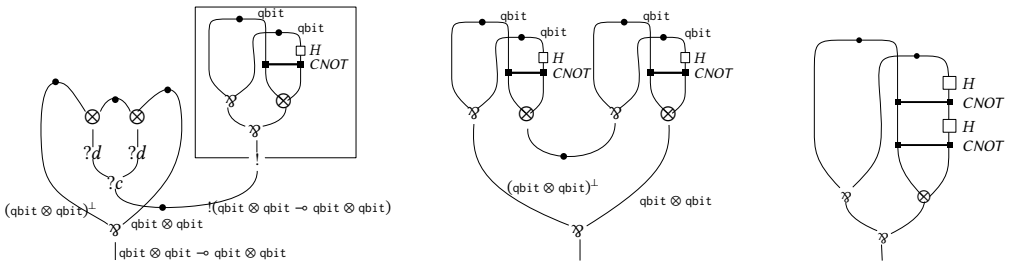


This might however lead to deadlock, as illustrated in the following (bogus) run



In the last panel, the +-box is unable to fire anything before the arrival of a token on its right input. But this token will never come since it would be resulting from the output of the same +-box.

Formally, dal Lago *et al.* [LFHY14] introduce proof-nets for SMLL, an extension of MLL with synchronization points. The authors describe a correctness criterion for ruling out deadlocks, and they present a token-based Geometry of Interaction where tokens flow from values to conclusions. They then sketch how this can be used to model a *strictly linear* quantum lambda calculus.



(a) Original (b) Intermediary state (c) Final Unfolding

Figure D.6: Instantiating and unrolling the net of Fig. D.3.

D.3.6 Towards a Quantum Geometry of Interaction

> I joined the research project at the time of the publication of [LFHY14], and my participation led to two publications [LFVY15, LFVY17]. In this section, I summarize the corresponding contributions.

The paper [LFVY15] presents a generalization of SMLL nets and their multi-token GoI to support exponential modalities and recursive behavior. The resulting nets —called SMEYLL[1]— therefore add to MELL-nets synchronization points and two additional boxes: the $\perp$-box of SMLL, to encode a primitive conditional, and the Y-box, to represent fixpoints. The presence of fixpoints forces us to consider a restricted notion of reduction, namely closed surface reduction (i.e., reduction never takes place inside a box). Cuts cannot be eliminated (in general) from SMEYLL proofs, as one expects in a system with fixpoints. Reduction, however, is proved to be deadlock-free, i.e., normal forms cannot contain surface cuts.

If we invite the reader to read the full paper for details [LFVY15], we present here a small example to illustrate the setting. Let us instantiate the term of Section D.3.2 to

$$(\lambda f.\lambda x.f(f\,x))(\lambda w.\texttt{let}\ \langle x,y\rangle = w\ \texttt{in}\ \mathit{CNOT}\langle x, H(y)\rangle)$$

The type A is $\texttt{qbit} \otimes \texttt{qbit}$, and the net is presented in Fig. D.6: Fig. D.6a and D.6b recall the original state and the result of the partial unfolding. Fig. D.6c shows the final result, and highlights an informal result: SMEYLL nets describe "folded" quantum circuits that the rewriting unfolds.

SMEYLL nets are seen as interactive objects through a synchronous interactive abstract machine (SIAM in the following). As for SMLL, multiple tokens circulate around the net simultaneously, and synchronize at sync nodes. In [LFVY15] however, SMEYLL nets and SIAM tokens do not support probabilistic behavior and can only carry very simple additional states such as natural numbers or Boolean values.

The follow-up paper [LFVY17] extends the setting to support quantum information and probabilistic side-effects. The resulting model then supports all of the structures needed to model the quantum lambda calculus. The model addresses two problems: the handling of entanglement, and the probability behavior.

The problem of entanglement can be exemplified by the following example: in the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, we want to be able to manipulate both qubits independently. In previous approaches [HH17, Del08b, DP08, Del08a], either this was not possible, or the expressive power of the system was too weak, lacking recursion and duplication [LZ14, LFHY14].

In the literature, probabilistic behavior is usually handled through sequentialization. This can be done with the help of a reduction strategy, as in the probabilistic lambda calculus

[1] We thought of using the name "SMELLY" but it was ruled out despite my heavy lobbying

[ETP14], or with the use of polarity [DH02, HMH14]. The proposal in [LFVY17] defines instead a confluent probabilistic transition system that is both infinitary and parallel.

Unlike the original proposal of [SV06], the new framework proposed in [LFVY17] models memory and choice effects in a parametric way, via a *memory structure*. The memory structure is presented in an algebraic manner and relies on nominal sets [Pit13]. This can be regarded as a generalization of Staton's equational system [Sta15], specific to quantum computation. Compared to purely categorical presentations of quantum GoI [HH17], this semantics offers a concrete, executable model where tokens follow the folded quantum circuit hidden inside a quantum program.

D.4 A Categorical Semantics for Circuit-Description

Up until this point, in the presentation there have been two distinct lines of research. On one hand, Chapter C discusses how quantum programming languages are all about circuit-description languages. On the other hand, this chapter has only been presenting semantics of quantum lambda calculi based on a very simple, QRAM operational semantics.

This section is devoted to closing the gap: we discuss a denotational semantics for a circuit-description lambda calculus. The challenge is that circuits are syntactic description: one cannot only rely on (some extension) of CPM. The usual suspect for sketching an answer is to rely on pure categorical constructs to capture all of the required structures of the language.

Section D.4.1 introduces the existing attempts at formalizing circuit-description languages with a focus on ProtoQuipper: a formal, core subset of Quipper expressed as an lambda calculus extension for circuit manipulation. Section D.4.2 sketches the proposal of Rennela and Staton [RS18a] for extending a concrete model of first-order quantum computation based on C^* algebras to express quantum circuit manipulation. However, because circuits are identified with the operations they represent, the semantics fails at encompassing syntactic circuit operations. Section D.4.3 discusses a purely categorical, more general semantics proposed by Selinger and Rios [RS18b]. It is still limited in the sense that it only supports a limited form of measurement. Finally, Section D.4.4 presents our proposal, answering this problem [LPVX21].

> The work presented in Section D.4.4 has been the result of the Ph.D thesis of 이동호 (Dongho Lee) [Lee22] who I co-supervised with Valentin Perrelle (CEA-LIST/LSL).

D.4.1 Formalizing Circuit-Description Languages

The formalization of circuit description languages started with two research threads. Arguably the first one is the formal language ProtoQuipper [Ros15], aimed at describing the core computational behavior of Quipper. The other approach has been the language QWIRE embedded in Coq [PRZ17], aiming at proving properties of quantum programs. Both works follow a similar approach to quantum programming: the circuit is a data-structure that is being constructed in a dynamic manner by a classical program. In the following I will concentrate on ProtoQuipper, as its structure is closer to the quantum lambda calculus already discussed in Section D.1.2. Moreover, ProtoQuipper has been the seminal work for many more improvements on the semantics side of quantum description languages [RS18b, LMZ18, FKS20, FKRS20, LPVX21, CD22, FKRS22a, FKRS22b].

ProtoQuipper can be regarded as an extension of the quantum lambda calculus. Instead of sending gates to the QRAM one at a time, the language features a constructor box for turning functions into circuits—i.e. buffering gates into an circuit-object that can then be manipulated as any other object. This box-operation can be regarded as a kind of thunk [Ing61, HD96] with partial evaluation [CD93]: a term box M will become a circuit-object, for instance a list of gates, but the gates will not be sent to the QRAM. In order to do so, another construct unbox aims at

"running" the circuit, effectively sending the gates downstream. Circuits are modeled in the language using a special arrow-type `circ`: a circuit with input A and output B is typed with $\texttt{circ}(A, B)$. We can therefore give the following type to box and unbox.

$$\begin{aligned} \text{box} &: !(A \multimap B) \multimap \texttt{circ}(A, B), \\ \text{unbox} &: \texttt{circ}(A, B) \multimap !(A \multimap B). \end{aligned}$$

The constant box makes a circuit out of the (partial evaluation) of a function, while box turns a circuit into a function.

In the original ProtoQuipper of Neil Ross [Ros15], the language would not support probabilistic behavior. So measurement is only allowed as a circuit gate sending a wire of type qubit to a wire of type bit. The possibility to turn a bit into a regular Boolean value on which to run the if-then-else construct of lambda calculus —the *dynamic lifting* feature— is not part of the formalism. It is then for instance not possible to realize dynamic circuits such as the one sketched in Figure C.1b.

Following Ross's work [Ros15], the team at Dalhousie has developed a categorical semantics for circuit-description languages [RS18b], based on the variant ProtoQuipperM. This language and its categorical semantics has been the seminal work on which most of the later works step up: [LMZ18] discusses (classical) recursion, [FKS20, FKRS20] generalizes the model to support dependent-types, while [FKRS22a, FKRS22b] (with ProtoQuipperDyn) and [LPVX21] (with ProtoQuipperL) study the addition of dynamic lifting to the language. The former approach [FKRS22a, FKRS22b] describes the set of axioms required for the categorical semantics to be sound, whereas the latter [LPVX21] constructs a concrete category based on quantum channels, and shows how the branching stemming from measurements can be seen as a Kleisli category in this framework (see Section D.4.4 for a more detailed discussion).

D.4.2 Semantics based on Operator Algebras

The semantics of regular, first-order quantum computation—with both unitaries and measurements—have been studied for a long time. If one trend of research focuses on mathematical, concrete models extending the original semantics of trace-non-increasing completely positive maps [Sel04a, Wes16, Wes19], other works follows a more axiomatic approach. Sam Staton [Sta15] in particular proposes a complete equational theory of first-order quantum computation, characterized by unitary applications and measurements. The equational theory is complete and comes with nine axioms, relying on C^* algebras: positive elements of C^* algebras can be regarded as observables in quantum theory. With Matthys Rennela [RS18a, RS20], they later explore how to build a linear-non-linear category à la Benton [Ben94a]. The model is very general and models any interacting computation involving a notion of circuit. To recover quantum computation (with measurement), they instantiate the model on Staton's equational theory [Sta15] (and C^*-algebras). As presented, the model is therefore very intentional: in its C^* instantiation, one can for instance hardly count the number of gates of a circuit within the model.

D.4.3 Semantics based on Category Theory

Following Ross's formalization of ProtoQuipper [Ros15], Rios and Selinger [RS18b] offer a categorical semantics of a related circuit-description language dubbed ProtoQuipperM. The semantics accounts for the box and unbox operations, as well as—when correctly instantiated—classical operators on circuits such as gate-count. The model is built from the following.

- A symmetric monoidal category M. Objects corresponds to bunches of wires and morphisms to circuits.

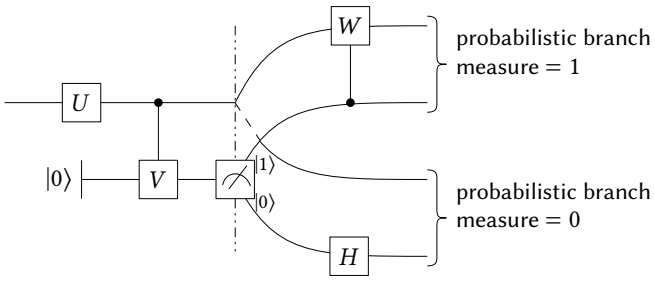


Figure D.7: Example of quantum channel.

- A symmetric monoidal closed category $\overline{M}$ with arbitrary products encapsulating M. This category is a technical vessel for the category $\overline{\overline{M}}$ defined next.
- A category $\overline{\overline{M}}$ aiming at modeling parameterized circuits: An object of $\overline{\overline{M}}$ is a pair $(X, \{A_x\}_{x \in X})$ where X is a set and A_x an object of $\overline{M}$. A morphism $(X, \{A_x\}_{x \in X}) \longrightarrow (Y, \{B_y\}_{y \in Y})$ is a pair $(f_0, \{f_x\}_{x \in X})$ where $f_0 : X \longrightarrow Y$ is a set-function and where for all $x \in X$, $f_x : A_x \longrightarrow B_{f_0(x)}$ is a morphism of $\overline{M}$.

One can canonically construct the embedding $p : \mathsf{Set} \longrightarrow \overline{\overline{M}}$, and p features an adjoint functor $\flat : \overline{\overline{M}} \longrightarrow \mathsf{Set}$. This adjunction describes a linear-non-linear category and turns $\overline{\overline{M}}$ into a model of linear logic [Ben94b]. Moreover, it makes it possible to model boxing and unboxing in a natural way: the homset $M(A, B)$ —the representation of a circuit between A and B— can be regarded as a map $A \multimap B$ in $\overline{\overline{M}}$.

If Rios and Selinger's construction can be extended to support recursive datatypes and fixpoints [LMZ18], it is *a priori* not expressive enough to support measurements as such. Indeed, if syntactic circuits can feature wires of type bit, this bit cannot be *lifted* to a Boolean value in the category $\overline{\overline{M}}$ of regular, classical computations.

D.4.4 Semantics for Circuits with Measurements

In [LPVX21], we propose the language ProtoQuipperL, extending ProtoQuipper with dynamic lifting. The box operation now not only captures unitary operations but also measurements, so that one can for instance box the function

$$\vdash \lambda x.\texttt{let}\ z = H\ (\texttt{qinit ff})\ \texttt{in if meas}\ z\ \texttt{then}\ U\ x\ \texttt{else}\ V\ x : \texttt{qbit} \longrightarrow \texttt{qbit}$$

(written in the language of Section D.1.2). Circuits are therefore now not only lists of gates, but *branching trees* accounting for the choice to make for continuing a circuit after a measure. Such generalized circuits are called *quantum channels*. A typical quantum channel is presented in Figure D.7: the measurement spawns two independent branches, one for each result of the measure. This follows the intuition drawn by the data-structure `CircIO` underlying Quipper's `Circ` monad presented in Section C.1.2.2.

We design a companion concrete category M of syntactic quantum channels that can account for (first-order) quantum computation with both unitaries and measurements, in the similar spirit as what was proposed by Ross Duncan [Dun09]. In particular, the category M is already monoidal closed and features products, so that we can identify $\overline{M}$ and M. We show how in this situation, the category $\overline{\overline{M}}$ features a monad capturing branching side-effects coming from the measure. This branching monad is the categorical interpretation of the `Circ` monad of Quipper. Quantum computations with dynamic lifting can then naturally be represented in the corresponding Kleisli category.

[SV09] Peter Selinger and Benoît Valiron. "Quantum lambda-calculus". In: [GM09]. Chap. 4, pp. 135–172.
[Val10a] Benoît Valiron. "Orthogonality and algebraic lambda-calculus". In: *Proceedings of the 7th International QPL Workshop Quantum Physics and Logic, QPL'10* (Oxford, UK). Ed. by Bob Coecke, Prakash Panangaden, and Peter Selinger. 2010, pp. 169–175. URL: http://www.cs.ox.ac.uk/people/bob.coecke/QPL_proceedings.html.
[Val12] Benoît Valiron. "Quantum computation: a tutorial". In: *New Generation Computing* 30.4 (2012), pp. 271–296. DOI: 10.1007/s00354-012-0401-7.
[Val13b] Benoît Valiron. "Quantum computation: from a programmer's perspective". In: *New Generation Computing* 31.1 (2013), pp. 1–26. DOI: 10.1007/s00354-012-0120-0.
[PSV14] Michele Pagani, Peter Selinger, and Benoît Valiron. "Applying quantitative semantics to higher-order quantum computing". In: [POPL14], pp. 647–658. DOI: 10.1145/2535838.2535879. arXiv: 1311.2290.
[VZ14a] Benoît Valiron and Steve Zdancewic. "Finite vector spaces as model of simply-typed lambda-calculi". In: *Proceedings of the 11th International Colloquium on Theoretical Aspects of Computing, ICTAC 2014* (Bucharest, Romania, Sept. 17–19, 2014). Ed. by Gabriel Ciobanu and Dominique Méry. Vol. 8687. Lecture Notes in Computer Science. See [VZ14b] for the long version. Springer, 2014, pp. 442–459. DOI: 10.1007/978-3-319-10882-7_26.
[VZ14b] Benoît Valiron and Steve Zdancewic. "Modeling simply-typed lambda calculi in the category of finite vector spaces". In: *Scientific Annals of Computer Science* 24.2 (2014), pp. 325–368. DOI: 10.7561/SACS.2014.2.325.
[LFVY15] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. "Parallelism and synchronization in an infinitary context". In: [LICS15], pp. 559–572. DOI: 10.1109/LICS.2015.58. HAL: hal-01231831. arXiv: 1505.03635.
[LFVY17] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. "The geometry of parallelism: classical, probabilistic, and quantum effects". In: [POPL17], pp. 833–845. DOI: 10.1145/3009837.3009859. HAL: hal-01474620. arXiv: 1610.09629.
[DGMV19] Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel, and Benoît Valiron. "Realizability in the unitary sphere". In: [LICS19], pp. 1–13. DOI: 10.1109/LICS.2019.8785834. HAL: hal-02175168. arXiv: 1904.08785.
[XVY21] Zhaowei Xu, Benoît Valiron, and Mingsheng Ying. "Reasoning about Recursive Quantum Programs". Draft, to appear in ACM TOCL. 2021. arXiv: 2107.11679.
[LPVX21] Dongho Lee, Valentin Perrelle, Benoît Valiron, and Zhaowei Xu. "Concrete categorical model of a quantum circuit description language with measurement". In: *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021*. Ed. by Mikolaj Bojanczyk and Chandra Chekuri. Vol. 213. LIPIcs. 2021, 51:1–51:20. DOI: 10.4230/LIPIcs.FSTTCS.2021.51.

Table D.8: Personal publications related to Chapter D.

Chapter E

Quantum Control and Reversible Computation

A typical execution in the coprocessor model consists of elementary gates applied to the quantum memory. The memory state consists of a superposition of basis elements: the gates are applied indistinctly on all basis elements at once. This model can be summarized by the slogan "quantum data, classical control" [Sel04a]. In this scheme, a quantum program is merely a classical program with classical control-flow, manipulating a quantum memory. The only thing in superposition in this model is the data.

However, a particular circuit combinator is hinting at a possibly finer execution model: the *control* of an operator. This combinator makes it possible to filter out the state space and only act on a subspace. A computational model of *purely quantum* executions generalizes the notion of controlled gate with quantum *superposition of executions* instead of only superposition of data [Nie97]. One shifts from a model of *classical control-flow* of programs to a *quantum control-flow*, where program (or circuits) can be superposed, yielding an alternative slogan:

> "Quantum data, quantum control".

This non-standard model of computation raises several challenges: this chapter discusses two of them. The first one is whether this model is realistic and can bring anything new compared to the standard circuit model. Another challenge of interest to us is the design of a suitable language to express superpositions of programs. In particular, the difficulty is to *preserve unitarity*.

- Section E.1 discusses the problem with the concrete implementation of quantum control. It discusses the literature on the subject and focuses on one of our main contributions: the quantum SWITCH [CDPV13]. This small protocol highlights how quantum control is not reducible to quantum circuits even though it was shown to be physically implementable. The section concludes by discussing the several approaches followed in the literature to define a syntax for superpositions of executions. In particular, the section discusses the notion of tests, loops, and recursion in a purely quantum context. It describes the problems that one must overcome while designing a syntax for quantum control.
- Section E.2 presents one of our contributions on the design of a syntax accounting for superpositions of programs. We focus on an extension of lambda calculus featuring terms in superposition, and we discuss the design of two possible type systems accounting for superpositions of terms and unitarity [ADV17, DGMV19]. In this approach, the language supports arbitrary linear combinations of terms. The type systems aims at determining which terms are "valid", i.e., make sense as quantum superpositions.

- Section E.3 presents our other contribution for a syntax of quantum control [SVV18]. Leaving the realm of pure lambda calculus, we propose a language based on pattern matching. The approach is dual to the one of Section E.2: Instead of allowing any linear combination of any programs, the syntax enforces valid quantum programs from first principles. We discuss how the language handles naturally both (some form of) recursion and unitarity. We also discuss how the corresponding type system agrees with an extension of linear logic: the logic µMALL [CSV23].

E.1 Implementing Quantum Control

Deciding to turn controlled gates into a general superposition of execution raises several questions: does it make sense in general? If yes, how does it differ from the regular model of quantum circuits? And, last but not least, how to *program* in this model? Each subsection addresses one question. Section E.1.1 discusses the debates pertaining to the physicality of quantum control. Section E.1.2 presents our seminal contribution on the topic: the quantum SWITCH. It consists in a minimal protocol exhibiting quantum superposition of execution. We show how the quantum SWITCH cannot be realized with quantum circuits. Section E.1.3 finally reviews the attempts at capturing quantum control within a syntax, and highlights the problems that occur.

E.1.1 Physicality of Quantum Control

This is part of a larger problem: the *physical Church-Turing thesis* whose scope is to describe what computation means within the constraints imposed by the laws of physics.

Citing Gandy [Gan80], the standard Church-Turing thesis[1] states that "Every effectively calculable function is a computable function". In the 1930s, on one hand, two main computational models were developed: "purely mechanical devices", the soon-to-become Turing machines [Tur36], and Church and Kleene's λ-definable functions [Kle35a, Kle35b, Chu36]. On the other hand, as described by Turing [Tur38], the notion of *effectively calculable* "refers to the intuitive idea without particular identification with any one of these definitions". Turing showed how these two definitions turn in fact out to be equal [Tur36], yielding the aforementioned thesis.

With its *physical Church-Turing thesis*, the problem unearthed by Gandy [Gan80] can be summarized by asking what physical process can be regarded as a valid "purely mechanical device". Unlike approaches attempting to describe the notion of computability from an axiomatic standpoint [DG08], Gandy derives a few physical principles entailing computational constraints on the behavior of any *reasonable* physical machine. His thesis "M" then states that "what can be calculated by a machine is computable" [Gan80].

Gandy was only considering the context of classical machines, leaving open the case of quantum computation. For the latter, Deutsch [Deu85] discussed its computational power and derived that it is no more powerful than classical computation: is the case closed? The question is not so clear. For instance, Nielsen [Nie97] discusses a paradox, with an (infinite-dimensional) unitary operator solving the halting problem: unitaries being the core elementary constructions for quantum computation, how to reconciliate the paradox with Deutsch's thesis [JS12a, AD12b]? According to Arrighi & Dowek [AD12b], the main problem lies with the infinite dimensionality that needs to be tamed: they provide a few physical principles, quantum equivalent to the one proposed by Gandy and ruling out Nielsen's paradox. This analysis sheds a new light on the physicality of non-standard models of quantum computation, such as quantum automata [Dow12, Arr19], and generally models based on *indefinite causal orders*: quantum causal graphs [AM17]; causal boxes [PMMRT17]; routed quantum circuits [VKB21];

[1] According to Copeland [Cop17], the term "Church-Turing thesis" was coined by Kleene [Kle67]

quantum switches [CDPV13, WC20]; supermaps [CDP08b, WDAB21]; extended circuit diagrams [LB21].

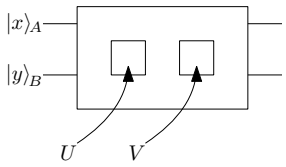
E.1.2 A Minimal Quantum Control: the Quantum SWITCH

One of the seminal works on *quantum control* and superposition of causal orders is [CDPV13], presenting the simplest example of *non-causal gate* ordering: the so-called *quantum SWITCH*. A presentation proceeds as follows. Suppose that you are given one *single* copy of a gate U and a gate V, and that you are asked to realize the operation $\textsc{Switch}(U, V)$ acting on two qubits A and B:

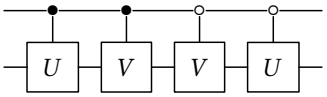
$$\begin{aligned} |0\rangle_A \otimes |y\rangle_B &\mapsto |0\rangle_A \otimes (UV\,|y\rangle_B), \\ |1\rangle_A \otimes |y\rangle_B &\mapsto |1\rangle_A \otimes (VU\,|y\rangle_B). \end{aligned} \tag{E.1}$$

Provided that U and V are unitary, it is easy to check that this 2-qubit operator is unitary. Depending on the state of the first qubit, the action on the second qubit is either UV or VU. But if the first qubit is in superposition, the action on the second qubit is *non-causal*.

Provided that U and V are known, fixed operators, $\textsc{Switch}(U, V)$ can be synthesized as a circuit without problem. The difficulty appears when U and V are unknown: we are therefore looking for a "higher-order" circuit with two holes such as



and whose behavior would be the one described in Eq. (E.1). As we showed in [CDPV13], such a "circuit with a hole", also known as *quantum comb* [CDP08a, CDP09], cannot possibly implement the behavior of Eq. (E.1). Of course, the quantum SWITCH can be realized if you ever had *two* copies of U and V —and if you were allowed to control unknown gates [FDDB14]— as follows:



But with only one copy of each, this is not possible.

Despite this impossibility within the circuit model, not only the quantum SWITCH has been shown to be physically realizable [PMAC+15, TCMG+21] but it has also been proven to bring a computational advantage [ACB14, TCMG+21] to be relevant in the context of quantum metrology [ZYC20] and thermodynamics [CVCC23].

E.1.3 Syntactic Approaches for Quantum Control

In order to analyze quantum superpositions of execution paths, another approach followed in the literature consists in focusing on the programming language constructs able to yield such a behavior. The focus is moved towards the control flow of a quantum computation happening *inside* the coprocessor: a *quantum control* flow instead of the (standard) *classical* control flow.

In conventional models of computation with algebraic effects such as non-deterministic or probabilistic computation, a successful approach has been to adapt the versatile lambda calculus to the new paradigm. The mainstream technique consists in encapsulating the side-effect inside a monad, following Moggi's proposal [Mog89]: the effect is *outside* of the calculus, only appearing as an epiphenomenon along the execution. A generic *computational* lambda calculus can be used for any side-effects representable with a monad. Another approach instead takes the algebraic effect as a *part of the computation*: the language is augmented with the

corresponding algebraic structure. In this paradigm, a term non-deterministically reducing to M or to N is typically represented with $M + N$ [dP95]. Similarly, one can equip the lambda calculus with a probabilistic sum [LZ12, Lev16], and more general cases can be handled by furthermore adding scalar multiplications [Vau09].

E.1.3.1 Van Tonder's Quantum Lambda Calculus

To make superpositions part of the computation, a natural solution therefore consists in considering *superpositions of lambda-terms*. The first author to try it out was van Tonder [Ton04] in 2004. His proposal is very intuitive: instead of coding lambda-terms on a regular memory, let us encode them on the *quantum* memory; the superposition therefore comes for free. In this language, one can for instance write $(\lambda x.H\,x)\frac{1}{\sqrt{2}}(0+1)$, represented in the memory as

$$|\langle\rangle \otimes |\lambda\rangle \otimes |x\rangle \otimes |.\rangle \otimes |H\rangle \otimes |x\rangle \otimes |\rangle\rangle \otimes \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle),$$

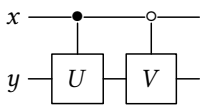
and meant to reduce to $|0\rangle$. The beta-reduction has to be encoded with quantum operations: typically with with a unitary map. In his paper, van Tonder describes the three main problems occurring along the way, and proposes solutions to them. The first problem is the fact that beta-reduction is not reversible: As for the case of reversible abstract machines [Klu99], this can be countered by keeping track of previous moves. The second problem is concerned with implicit weakening. Consider as an illustration the term $|\lambda x.0\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$: a naive interpretation yields $\frac{2}{\sqrt{2}}|0\rangle$, whose norm is not 1. The problem comes from the non-linearity of the lambda-term. Finally, the third problem occurs with non-trivial terms in superpositions. Consider for instance $\frac{1}{\sqrt{2}}(|(\lambda xy.x)\,1\,0\rangle + |(\lambda xy.y)\,0\,1\rangle)$: although this state is arguably of norm 1, it should reduce to a state of norm different from 1. Van Tonder then describes his solution, which is to restrict the language to a system where terms in superposition have to be equal, apart for 0's and 1's: this essentially amounts to only having classical control, as this corresponds precisely to the quantum lambda calculus of [SV06]. In other words, this simple, naive approach fails to capture any quantum control.

E.1.3.2 QML

The first successful attempt at quantum control can be traced back to QML [AG05a, AG05b, AGVS05]: in this line of work, the authors present the first example of a programmable *quantum test*, together with a compiler to circuits. In QML, it is possible to give a formal meaning to the intuitive program

$$x : \mathtt{qbit}, y : \mathtt{qbit} \vdash \mathtt{if}^\circ\ x\ \mathtt{then}\ \langle x, U\,y\rangle\ \mathtt{else}\ \langle x, V\,y\rangle : \mathtt{qbit} \otimes \mathtt{qbit},$$

which inputs two qubit wires x, y and compiles down to the circuit



The $\mathtt{if}^\circ$-construct consists in a *quantum* test: unlike the if-then-else construct in the quantum lambda calculus presented in Section D.1.2, the qubit x is not measured, and both branches happen in parallel. For this to make sense, we however need both branches to somehow yield orthogonal states. For instance, the term

$$x : \mathtt{qbit}, y : \mathtt{qbit} \vdash \mathtt{if}^\circ\ x\ \mathtt{then}\ x\ \mathtt{else}\ \mathit{NOT}\,x : \mathtt{qbit}$$

is not valid, since it maps $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ to $\frac{2}{\sqrt{2}}|1\rangle$, therefore not preserving the norm.

Altenkirch & Grattage proposes a small, first-order language with a simple type system of tensors of qubits. The system comes with a syntactic notion of orthogonality, but, partly because of the limited expressiveness of the type system it is very constrained. Despite its limitations, it does compile to quantum circuits: it is therefore "fully quantum".

E.1.3.3 Linear Algebraic Lambda Calculus

QML answers one of the problems of van Tonder's quantum lambda calculus: superposing distinct execution flows, but at the expense of expressiveness. An alternative to gain expressive power is to lift part of the restrictions imposed by the encoding onto the quantum memory.

Instead of requiring a strict unitarity of the beta-reduction while asking for a norm condition on terms, *Lineal*, the *linear, algebraic lambda calculus* of Arrighi & Dowek [AD08, AD17] support *any* linear combination of terms:

$$M, N \quad ::= \quad x \mid \lambda x.M \mid MN \mid \alpha \cdot M \mid M + N \mid \vec{0}$$

This line of work questions the fundamental notion of computation: what does it mean to *compute in a vector space* [AD05]?

The operational semantics of Lineal formalizes the idea of "terms-as-operators": a (pure) term is seen as a basis vector. The application is then distributive over sum and scalar multiplication:

$$\begin{aligned}(\alpha_1 \cdot M_1 + \alpha_2 \cdot M_2)(\beta_1 \cdot N_1 + \beta_2 \cdot N_2) \to^* \\ \alpha_1\beta_1 \cdot (M_1N_1) + \alpha_1\beta_2 \cdot (M_1N_2) + \alpha_2\beta_1 \cdot (M_2N_1) + \alpha_2\beta_2 \cdot (M_2N_2),\end{aligned} \tag{E.2}$$

while the λ-constructor is not, acting like a *thunk* [Ing61, HD96]. Formally, the beta-reduction is extended with a set of rules for manipulating sum and scalar multiple of terms, such that Eq. (E.2) can be deduced. Several possibilities exists: are terms considered modulo associativity and commutativity? modulo the algebraic equational theory? Each of these possibilities provide sensible —and related— models of computation [DPTV10, ADPTV14]: as shown in [AD05], the equational theory for vector spaces can be made into a confluent rewrite system.

Lineal follows a call-by-value reduction strategy. Or, more precisely, a *call-by-base* reduction strategy: $(\lambda x.M)V$ only evaluates whenever V is a *pure* term: a term that is not a distribution.

The underlying idea is that terms are regarded as generalized operators. In particular, it is then possible to encode matrices:

$$U \quad \triangleq \quad \begin{pmatrix} \alpha & \beta \\ \delta & \gamma \end{pmatrix}$$

can be regarded as a map acting on the space of terms generated by $\mathtt{t} \triangleq \lambda xy.x$ and $\mathtt{f} \triangleq \lambda xy.y$. The matrix U can be modeled with

$$M_U \quad \triangleq \quad \lambda b.b\,(\lambda z.(\alpha \cdot \mathtt{t} + \delta \cdot \mathtt{f}))(\lambda z.(\beta \cdot \mathtt{t} + \gamma \cdot \mathtt{f}))\lambda z.z.$$

The term $M_U\,(a \cdot \mathtt{t} + b \cdot \mathtt{f})$ then reduces to $(a\alpha + b\beta) \cdot \mathtt{t} + (a\delta + b\gamma) \cdot \mathtt{f}$, corresponding to the operation

$$\begin{pmatrix} \alpha & \beta \\ \delta & \gamma \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}.$$

As Arrighi & Dowek discuss, in Lineal one can encode matrices, vectors, but also tensor products, and therefore simulate quantum circuits.

If this extension of lambda calculus looks promising, it is however to take with care. Indeed, consider the following term

$$Y_M \quad \triangleq \quad (\lambda x.(xx + M))(\lambda x.(xx + M)). \tag{E.3}$$

The term Y_M reduces to $Y_M + M$. But then $Y_M - Y_M$ is beta-equivalent to both M and 0, the null vector: all terms collapse to zero.

Arrighi & Dowek address the problem by by enforcing a rewriting strategy disallowing the reduction of terms such as $Y_M - Y_M$. This consistency problem however appears in many algebraic extensions of lambda calculi, and several approaches to deal with the problem have been proposed in the literature [Vau09, Val10b, Val13a].

E.1.3.4 Other Algebraic Extensions of Lambda Calculus

Lineal fits within the large class of *algebraic extensions of lambda calculus*. The origin of the study can be traced back to Breazu-Tannen, discussing code optimization at compilation time [BM87], and how replacing `x - x` with `0` can be problematic within an untyped setting. This seminal paper yielded a line of works showing how type discipline can help [Bre88], and how the consistency of the system is related to the confluence of the underlying algebraic rewrite system, whether in a typed [BG89, BG91, BG94] or in an untyped setting [Dou92].

Algebraic lambda calculi in the style of Lineal —that is, where linear combinations of terms are themselves terms— were introduced concurrently to Arrighi & Dowek [AD08] within the context of the differential lambda calculus [ER03], stemming from an analysis of quantitative models of linear logic [Ehr02]. The interaction of the algebraic structure and the lambda calculus in this context has then been studied by Vaux [Vau09], who rediscovered the problems discussed by Breazu-Tannen 20 years earlier [BM87]. With linear combinations, the algebraic structure of terms is very rigid, and Vaux discusses several ways to recover consistency: with a type system enforcing strong normalization of terms, or with positive scalars (thus ruling out terms such as the one of Eq. (E.3)), or with finitely splittable scalars.

Compared to Arrighi & Dowek approach, Vaux's algebraic lambda calculus [Vau09] is *call-by-name*: application is not distributive on the right, so

$$(\lambda x.M)(\alpha \cdot N_1 + \beta \cdot N_2) \to M[x := \alpha \cdot N_1 + \beta \cdot N_2]$$

while

$$M(\alpha \cdot N_1 + \beta \cdot N_2) \quad \neq \quad \alpha \cdot MN_1 + \beta \cdot MN_2.$$

The two are incompatible, as it is already the case, say, in probabilistic computation: tossing a coin and duplicating the result is not the same thing as tossing twice the coin.

E.2 Typing Superpositions of Lambda-Terms

This section presents our work on the development of a type system for the lambda calculus presented in Section E.1.3.3. Extended with linear combinations of terms, this lambda calculus aims at modeling quantum superposition of programs.

The challenge addressed in this section concerns the validity of a lambda-term in superposition. How can we decide whether such a program indeed represents a physical, quantum operation? We want for instance a program to correspond to a unitary operation.

One of the formal tools to separate "valid" programs from "invalid" ones is the use of a type system. It consists in a formal term annotation, stable under composition, and characterizing a property we want "valid" programs to satisfy. Typical use-cases for a type system are termination and error-freeness.

This section presents two type systems for a linear algebraic lambda calculus. The type system of Section E.2.1 comes as a set of sophisticated, compositional definitions. Proving properties of well-typed terms then requires complex proofs. The type system presented in Section E.2.2 is instead defined organically using the operational semantics: a type is a set of terms with suitable properties. The compositionality of the type system is derived as a corollary. We discuss how the system we obtain is more fine-grained.

Both of the works presented here are collaborations with Alejandro Díaz Caro. The one discussed in Section E.2.1 started while Alejandro was doing his Ph.D—we are still collaborators nowadays. In Section E.2.2, I present a work Alejandro and I realized later on, on a collaboration with the logic group in Montevideo (Uruguay).

E.2.1 An Axiomatic Type System: Vectorial System-F

As discussed in Section E.1.3.4, the simplest strategy to recover consistency for an algebraic lambda calculus is to add a type system enforcing termination.

Vaux's simple type system is very natural: it consists in typing $\sum_i \alpha_i \cdot M_i$ with A as long as each M_i can be typed with A. This approach is akin to the approach one can follow in the context of probabilistic or non-deterministic behavior: terms "in superpositions" should share the same type A, and the overall "computation" is then given the type A.

Instead, the approach we followed in [ADV17] is to allow terms with distinct types to be summed. This section describes the approach.

E.2.1.1 Simply-Typed Vectorial Lambda Calculus

The grammar for the vectorial lambda calculus is the same as the one of Lineal, presented in Section E.1.3.3. A simple type system is as follows:

$$A, B \quad ::= \quad X \mid A \Rightarrow B.$$

Following the Lineal approach, the rewrite-system of the vectorial lambda calculus is *call-by-base*.

Coding Qubits. In the regular lambda calculus the Boolean values $\mathtt{tt}$ and $\mathtt{ff}$ can be coded with $\lambda xy.x$ and $\lambda xy.y$. These terms can both be typed with $X \Rightarrow X \Rightarrow X$. Within the vectorial lambda calculus, it is possible to write any linear combination

$$\alpha \cdot \lambda xy.x + \beta \cdot \lambda xy.y,$$

and the typing rules can give to all of these terms the type $X \Rightarrow X \Rightarrow X$. If scalars range over the complex field, and if we impose $|\alpha|^2 + |\beta|^2 = 1$, we can claim to have embedded the state of quantum bits in the vectorial lambda calculus.

Quantum If. With the Boolean values coded as $\lambda xy.x$ and $\lambda xy.y$, in the regular lambda calculus, the if-then-else construct $\mathtt{if}\, M\, \mathtt{then}\, N\, \mathtt{else}\, P$ can simply be written with $(MN)P$. As we are in a call-by-value setting, we want to forbid the branches of the test to evaluate: we can use *thunks* [Ing61, HD96] to "freeze" the computations in the branches, as follows:

$$\mathtt{if}\, M\, \mathtt{then}\, N\, \mathtt{else}\, P \quad \triangleq \quad ((M(\lambda z.N))(\lambda z.P))\, * \tag{E.4}$$

where z is a fresh variable and $*$ is any closed normal form, for instance $\lambda x.x$.

The vectorial lambda calculus being call-by-value, we can rely on the encoding of Eq. (E.4) to emulate the behavior of a "quantum test" as in QML. As the language does not enforce any unitary constraint, we can in fact encode *any* matrix. Consider for instance the map U sending $\mathtt{tt}$ to $\alpha \cdot \mathtt{tt} + \beta \cdot \mathtt{ff}$ and $\mathtt{ff}$ to $\gamma \cdot \mathtt{tt} + \delta \cdot \mathtt{ff}$. The operator U can be emulated with the term

$$U \quad \triangleq \quad \lambda x.\mathtt{if}\, x\, \mathtt{then}\, (\alpha \cdot \mathtt{tt} + \beta \cdot \mathtt{ff})\, \mathtt{else}\, (\gamma \cdot \mathtt{tt} + \delta \cdot \mathtt{ff}) \tag{E.5}$$

using the encoding of Eq. (E.4).

Typing the operator U with the simple type system is akin to typing if-then-else in the regular simply-typed lambda calculus: to get a portable solution the type system misses universal

quantifiers. Such quantifiers can be *à priori* easily added to the vectorial lambda calculus, with the two standard typing rules

$$\frac{\Delta \vdash M : \forall X.A}{\Delta \vdash M : A[X := B]} \qquad \frac{\Delta \vdash M : A \quad X \notin \mathsf{FV}(\Delta)}{\Delta \vdash M : \forall X.A}$$

The Boolean values (and their linear combinations) can then be typed with $\forall X.X \Rightarrow X \Rightarrow X$, and the operator U with

$$(\forall X.X \Rightarrow X \Rightarrow X) \Rightarrow (\forall X.X \Rightarrow X \Rightarrow X).$$

However, as expressive as it is this type system is unable to capture algebraic properties of terms. Several studies have in particular been performed by Arrighi & Díaz-Caro [AD09, DD17, AD12a, ADV11, ADV17], with the addition of scalars or more generally a vectorial structure to types.

E.2.1.2 Quantifiers: Vectorial System-F

The objective of this section is to present the work initiated in [ADV11] and achieved in [ADV17]. Its aim is to capture some algebraic properties of the vectorial lambda calculus within a type system. Schematically, if $M : A$ and $N : B$, we aim at a meaningful way of saying that $\alpha \cdot M + \beta \cdot N$ is of type $\alpha \cdot A + \beta \cdot B$. The type-system should also be expressive enough to be able to give a parametric type to the operator U presented in Eq (E.5). This was conceived as a first step towards a type system enforcing unitarity constraints.

The language of Section E.2.1.1 is now equipped with the type grammar

$$\begin{array}{lllll} \text{Types} & T, R, S & ::= & U \mid \alpha \cdot T \mid T + R \mid \mathbb{X}, \\ \text{Unit Types} & U, V, W & ::= & \mathcal{X} \mid U \Rightarrow T \mid \forall \mathcal{X}.U \mid \forall \mathbb{X}.U. \end{array}$$

The type system acknowledges the fact that any term is first and foremost a linear combination of *base terms*: a general type is therefore a linear combination of *unit types*, where pure types are meant to type *base terms*. A base type cannot be a linear combination: it is therefore either an arrow-type or a quantified type. The type system features two kinds of type variables: type variables $\mathbb{X}$ standing for general types, and type variables $\mathcal{X}$ standing for unit types. Arrow types reflect the fact that the language is call-by-base: the domain of an arrow type is a unit-type. Indeed, consider $\lambda x.M$: although M can be any term, the term variable x can only be replaced by a base term, and base terms are meant to be typed with unit types.

The types come equipped with an equivalence relation $\equiv$, defined as follows:

$$\begin{array}{cc} 1 \cdot T \equiv T & \alpha \cdot T + \beta \cdot T \equiv (\alpha + \beta) \cdot T \\ \alpha \cdot (\beta \cdot T) \equiv (\alpha\beta) \cdot T & T + R \equiv R + T \\ \alpha \cdot T + \alpha \cdot R \equiv \alpha \cdot (T + R) & T + (R + S) \equiv (T + R) + S \end{array}$$

This relation is used in the typing rules for the algebraic aspect of the language, as follows:

$$\frac{\Gamma \vdash M : T}{\Gamma \vdash \alpha \cdot M : \alpha \cdot T}\ (\alpha_I) \qquad \frac{\Gamma \vdash M : R \quad \Gamma \vdash N : T}{\Gamma \vdash M + N : R + T}\ (+_I) \qquad \frac{\Gamma \vdash M : R \quad R \equiv T}{\Gamma \vdash M : T}\ (\equiv)$$

Finally, the term 0 can be given any *inhabitated* type, as follows:

$$\frac{\Gamma \vdash M : T}{\Gamma \vdash 0 : 0 \cdot T}\ (0_I)$$

This rules out the possibility to introduce bogus, empty types inside a linear combination. It stems from the fact that the only possible way to introduce a 0-term is through the rewrite rule $0 \cdot M \longrightarrow 0$.

One peculiar thing to note in the type system is the absence of a 0-type: there is no notion of "empty" linear combination of types. One of the reasons is consistency: With a 0-type, it would make sense to ask for $0 \cdot T \equiv 0$, thus rendering all 0-scalared types equal, including empty ones.

Dealing with functions. The typing rule for the lambda-abstraction follows the intuition given while describing the type system:

$$\frac{\Gamma, x : U \vdash M : R}{\Gamma \vdash \lambda x.M : U \Rightarrow R} \ (\Rightarrow_I)$$

The rule for the application is more involved, as we want to be able to handle for instance terms of the form $(M_1 + M_2)(N_1 + N_2)$. For the purpose of the discussion, we only give an example and we invite the reader to consult the full paper [ADV17] for details.

Because quantifiers can only happen at the level of unit-types, the typing rule for application contains both an arrow-elimination and a quantifier-elimination. In order to illustrate what we mean, assume that we have defined pairs and projections in the usual way, using the second-order. Now, the term $(\pi_1 + \pi_2)(\langle U_1, U_2\rangle + \langle V_1, V_2\rangle)$ reduces to $U_1 + U_2 + V_1 + V_2$: assuming that the U_i's and V_i's are well-typed, this should also be well-typed. Following the way the rewrite procedure distributes the application over the sum, the rule can be stated as

$$\frac{\Gamma \vdash M : \forall \mathcal{X}\mathcal{Y}.(\mathcal{X} \times \mathcal{Y} \Rightarrow \mathcal{X}) + \forall \mathcal{X}\mathcal{Y}.(\mathcal{X} \times \mathcal{Y} \Rightarrow \mathcal{Y}) \quad \Gamma \vdash N : (U_1 \times U_2) + (V_1 \times V_2)}{\Gamma \vdash MN : U_1 + U_2 + V_1 + V_2.}$$

Both of the types $U_1 \times U_2$ and $V_1 \times V_2$ are matched against the domain $\mathcal{X} \times \mathcal{Y}$ of the function, and in each case each summand of the type of the function yields its part, therefore producing all of the U_i's and V_j's.

This instance of the application rule features all of the subtleties of the general version presented in [ADV17].

Discussion. The vectorial lambda calculus can emulate (finite-dimensional) linear operations: one can encode quantum circuits in the language. The proposed vectorial System-F is then expressive enough to correctly type this encoding. Moreover, the type system enforces consistency: all typed terms are strongly normalizing. However, the standard property of subject reduction, stating that if $M \to N$ and $M : A$ then $N : A$ does not quite hold in our system. The problem comes from the mismatch between the equivalence on types that does not equate T and $T + 0 \cdot R$ and the rewrite system, sending all terms of the form $0 \cdot M$ to 0. The language however features a weakened version, based on a relation $\sqsupseteq$ satisfying several rules, among which $T \sqsupseteq T + 0 \cdot R$, capturing the "problem" with the zero-term.

E.2.2 A Type System Based on Realizability

Two strategies can be followed in order to design a type system. On one hand, one can define a formal grammar of types, and types can then be attached to terms using an (external) set of axiom rules. This is what has been done in Section E.2.1 A type system usually aims at capturing various properties of typed terms: These properties are then proven from the typing rules, as corollaries. On the other hand, one can follow the route of *realizability* [Kle45] and instead define types inductively, as *sets of terms*. The properties to enforce on typed terms can be added as constraints on the definition of the types. In this situation, typing rules becomes lemmas to be proven, instead of primitive axioms. The fact that a well-typed term verifies one of desired property is obtained "by definition".

E.2.2.1 Example based on simply-typed lambda calculus

The archetypal example is the proof of strong normalization of a typed lambda calculus. On one hand, one can set up the type up front with an abstract grammar:

$$\begin{array}{lcl} M, M & ::= & x \mid \lambda x.M \mid MN, \\ A, B & ::= & \sigma \mid A \Rightarrow B, \end{array}$$

set up a notion of typing judgment $x_1 : A_1, \dots, x_n : A_n \vdash M : B$ and define what it means to be a valid typing judgment with a series of typing rules, posed as axioms:

$$\frac{}{\Delta, x : A \vdash x : A} \qquad \frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x.M : A \Rightarrow B} \qquad \frac{\Delta \vdash M : A \Rightarrow B \quad \Delta \vdash N : A}{\Delta \vdash MN : B} \tag{E.6}$$

The rewrite system of lambda calculus is based on beta-reduction: $M \to N$ is defined as the smallest congruent relation satisfying $(\lambda x.M)N \to M[x \mapsto N]$. It is well-known that although the untyped lambda calculus is not strongly normalizing, the simply-typed fragment is. A proof can be designed using *reducibility candidates* [Tai67, GLT90].

The intuition behind reducibility candidates consists in defining for each type A a set of terms RED_A, called *reducibility candidates of type* A. They are defined by induction, following the "structure" of the types. For instance, $M \in \mathrm{RED}_{A \Rightarrow B}$ whenever for all $N \in \mathrm{RED}_A$, we have $MN \in \mathrm{RED}_B$. For the base case RED_σ, we enforce the desired property: $M \in \mathrm{RED}_\sigma$ whenever it is strongly normalizing and of type σ. One then derives various properties of these sets of terms, from which one can conclude strong normalization of well-typed terms.

An alternative approach to typing consists in directly starting from the computational behavior given by the beta-reduction. In this setting, we start from an untyped lambda calculus, and we define types in a semantic manner as *closed normal forms*. A term M is a *realizer* for a type A, denoted with $M \Vdash A$, if M reduces to a term in A. We then define the arrow $(\Rightarrow)$ as an operator on sets of terms as follows:

$$X \Rightarrow Y \quad \triangleq \quad \{\ \lambda x.M \text{ closed term} \mid \forall N \in X,\ M[x \mapsto N] \Vdash Y\ \}. \tag{E.7}$$

Note how realizers of $X \Rightarrow Y$ are strongly normalizing when the realizers of X and Y are strongly normalizing. Typing judgments of the form $x_1 : A_1, \dots, x_n : A_n \vdash M : B$ are then defined as a shortcut notation for

$$\forall N_1 \in A_1, \dots N_n \in A_n,\ \ M[x_1 \mapsto N_1, \dots, x_n \mapsto N_n] \Vdash B.$$

The 3 typing rules of Eq. (E.6) then become lemmas: we essentially have to choose as base type σ a set of strongly normalizing closed terms…

The system is very versatile: it can be applied to extended lambda calculi, and other type operators can be constructed from their computational interpretation: products, co-products, lists, quantifiers, *etc* [Lep16].

E.2.2.2 Weak Vector Spaces

A realizability model heavily relies on the *computational behavior* of an *untyped* language. As discussed in Section E.1.3.3, without types the consistency of the vectorial lambda calculus is not guaranteed, seemingly jeopardizing any meaningful notion of normal form and, therefore, computation.

In a research thread [Val10b, Val13a], we propose a solution to this problem. The idea is akin to what was proposed for the algebraic structure of the vectorial system-F: disconnect the 0-vector and vectors scaled to 0: the latter register what information got zero-ed out while in the former case everything is discarded. When the information is consistent one can indeed identify both approaches — but this is not possible anymore if the information is inconsistent, as for instance with the term Y_M of Eq. (E.3).

The solution we propose for retaining consistency in a vectorial lambda calculus where arbitrary fixpoints are allowed is to *weaken* the equations of module by disallowing the rule equating $0 \cdot M$ and $\vec{0}$: if we still have $Y_M - Y_M = 0 \cdot Y_M$, one cannot anymore get to $\vec{0}$. Formally, if $(\mathcal{A}, 1, \star, 0, +)$ is a ring, an *weak $\mathcal{A}$-module* $(M, +, \vec{0}, \cdot)$ is the data consisting of a commutative monoid $(M, +, \vec{0})$ and an operation $(\cdot) : \mathcal{A} \times M \to M$ such that for all $a, b \in \mathcal{A}$ and for all $x, y \in M$,

$$\begin{aligned} a \cdot (x + y) &= a \cdot x + a \cdot y, \quad & a \cdot (b \cdot x) &= (a \star b) \cdot x, \\ (a + b) \cdot x &= a \cdot x + b \cdot x, & 1 \cdot x &= x. \end{aligned}$$

In particular, we do *not* impose $0 \cdot x = \vec{0}$, meaning that $\vec{0}$ is not the same as $x + (-1) \cdot x$.

By turning the vectorial lambda calculus into a *weak* module, the term $Y_M - Y_M$ still exists but cannot be used to collapse the equational theory anymore: it does not equate $\vec{0}$ anymore. This has been formalized in [Val13a], where I show how a typed vectorial lambda calculus with fixpoint based on an equational theory of weak module admits a non-trivial model.

E.2.2.3 Realizability Model Capturing Unitary

Based on [Val13a], it is possible to give a sound computational interpretation of a vectorial lambda calculus (based on weak modules). With such a computational interpretation, one can then design a realizability model. Interestingly enough, it is even possible to capture *within a type system* a notion of unitarity: this is the topic of a collaboration with Alejandro Díaz Caro, Mauricio Guillermo and Alexandre Miquel [DGMV19].

The language we consider is a (vectorial) lambda calculus extended with constructs to deal with a unit term $\star$, pairing and (binary) injections. For the purpose of the discussion, we note the pairing of M and N as $\langle M, N \rangle$, and we consider Boolean values $\mathtt{tt}$ and $\mathtt{ff}$ defined in the usual way using injections. The pairing is bilinear with respect to the weak module structure, so for instance we have

$$\langle M_1 + M_2, \alpha \cdot N \rangle = \alpha \cdot \langle M_1, N \rangle + \alpha \cdot \langle M_2, N \rangle.$$

The operational semantics is "call-by-base": one does not reduce under lambda-abstractions.

Notation E.1. For this section, we use the following notation: V, W stands for pure values: lambda-abstractions, pairs of values, or injections of values ; $\vec{V}, \vec{W}$ stands for linear combinations of pure values, and M, N stands for general terms.

Since the ring is the field of complex numbers, we show in the paper that some of the notion of Hilbert spaces can be defined in this weakened context. In particular, one can define the naive notion of scalar product and ℓ^2-norm as follows:

$$\begin{array}{ll} \left\langle \sum_i \alpha_i \cdot V_i \,\middle|\, \sum_j \beta_j \cdot W_j \right\rangle & = \sum_{i,j} \overline{\alpha_i} \beta_j \cdot \delta_{V_i, W_j} \\ || \sum_i \alpha_i \cdot V_i || & = \sqrt{\sum_i |\alpha_i|^2} \end{array}$$

where the V_i's and the W_j's are pure values and $\delta_{V,W} = 1$ if $V = W$ and 0 otherwise[2]. This gives a notion of orthogonality, making for instance $\mathtt{tt}$ and $\mathtt{ff}$ orthogonal. We can define the *span* $\mathrm{span}(X)$ and the *basis* $\flat X$ of a set of terms X

$$\begin{array}{ll} \mathrm{span}(X) & = \left\{ \sum_{i=1}^n \alpha_i \cdot M_i \;\middle|\; n \in \mathbb{N},\ \forall i, M_i \in X \right\}, \\ \flat X & = \left\{ V \mid \alpha \cdot V + M \in X \right\}. \end{array}$$

We can also define the *unit sphere* of values as

$$S_1 = \left\{ \vec{V} \;\middle|\; ||\vec{V}|| = 1 \right\}.$$

This gives a canonical notion of "normalized vector" in the "vector space" of linear combinations of terms.

In this model, the type of Boolean values can then be defined as $\mathbb{B} \triangleq \{ \mathtt{tt}, \mathtt{ff} \}$, and the type of quantum bits as $\mathbb{Q} \triangleq \mathrm{span}(\mathbb{B}) \cap S_1$. A qubit is then literally a superposition of Boolean values.

In general, we define a *unitary type* as a subset of S_1: In [DGMV19], on top of $\flat$ we define several unitary type operators, among which:

$$A \times B = \left\{ \langle \vec{V}, \vec{W} \rangle \,\middle|\, \vec{V} \in A, \vec{W} \in B \right\}, \qquad A \otimes B = \mathrm{span}(A \times B) \cup S_1.$$

[2] There is some subtlety in term equality. Please refer to the paper [DGMV19] for details

Note how $\mathbb{Q} \otimes \mathbb{Q}$ consists in normalized linear distributions of pairs of Boolean values.

We can define function types as discussed for Eq. (E.7), and from the system we can derive that the function type $\mathbb{Q} \to \mathbb{Q}$ corresponds to unitary operators on the Hilbert space $\mathbb{C}^2$. We then derive a set of typing rules [DGMV19, Tab. 6], providing a compositional way to construct terms that enforces unitarity "by construction".

Discussion. In order to assess the expressiveness (and the versatility) of the language, we show how to embed the quantum lambda calculus (albeit without measurement), similar to the one discussed in Section D.1.2. Moreover, because of vectorial structure of the language, we show that we can also define a control operator, and, in general define a form of quantum SWITCH (discussed in Chapter E), thus offering a framework for both classical and quantum control.

However, although this paper offers a solution to the long-standing question of seeing vectorial lambda calculus as a medium for representing quantum computation, it is still unsatisfactory. The main limitation stands in the discrepancy between the *semantic* interpretation of unitary terms in the realizability model and their concrete, syntactic structure. One interesting research direction consists in developing a compilation framework to turn valid terms into executable quantum circuits.

E.3 Reversible and Quantum Pattern-Matching

The vectorial lambda calculus is based on a inherently irreversible model of computation. Instead of trying to adjust its defects to purely quantum computation, an alternative approach consists in changing the paradigm and moving towards *reversible computation*.

This section is devoted to a model of computation alternative to lambda calculus, reversible and based on pattern-matching [SVV18]. This computational model shares links with the vectorial lambda calculus, yet it allows a finer control over problematic aspects: there is no need for weak modules, yet the system supports a notion of recursive behavior.

Section E.3.1 presents the concept of reversible language, while Section E.3.2 discusses how reversible pattern-matching can be seen as a primitive design for a reversible language. Section E.3.3 sketches its categorical interpretation, while Section E.3.4 provides an strategy for introducing inductive types and recursion. Section E.3.5 discusses how to extend the language to quantum control, and Section E.3.6 reviews a Curry-Howards interpretation based on the logic µMALL.

> If this line of work started from an epistolary discussion with Amr Sabry and Juliana Vizzotto, it has become one of my main research vehicles in recent years. In particular, reversible pattern matching has been the subject of the Ph.D of my students Kostia Chardonnet [Cha23], co-supervised with Alexis Saurin (IRIF) and Louis Lemonnier [Lem24], co-supervised with Vladimir Zamdzhiev.

E.3.1 Background on Reversible Language

Discussed in Section C.2.1, the subject of reversible computation has spurred an avenue of research in programming languages and type systems.

On one hand, following the trend of research oriented towards reversible architectures [FBCH+20], a line of research aims at designing imperative programming languages targeting reversible computation. The ancestor of such languages is arguably Janus [Lut86], an imperative flow-chart language rediscovered in [YG07] and studied in details in [YAG16]. Reversible computation is linked to linearity, and [Bak92, Mat03] explore the subject with respectively Ψ-lisp and SRL, akin to a linear Janus language.

In the realm of syntactic functional languages, lambda calculus is not well-suited — although there has been a proposal for a reversible combinatory algebra [PHW06]. In terms of syntax, a seminal proposal for a reversible, functional language is the untyped Rfun [YAG12] — although Thomsen concurrently aimed at a proposal [Tho12]. From this seminal Rfun several extensions were developed: heap manipulation and algebraic datatypes in [AG13], the ability to manipulate non-linear objects in [Mog14], the addition of garbage collection in [Mog18]. On the side of concrete use of the language, a series of examples of code have been proposed [TA15], while on the formal side a core fragment of the language (CoreFun) has been analyzed in [JKT18].

A last trend consists in the study of point-free languages. The first approach is the language INV [MHT04], followed by Π, presented by James and Sabry [JS12b] with a sound presentation of isomorphism between types and a discussion on irreversibility as side effect. A (reversible) compiler for the language has also been designed [JS12c]. A specific discussion on type systems for reversible programming as semirings can be found in [CS16].

To conclude, an approach linking point-free languages and regular, typed functional language is Theseus [JS14]. Initiated in [SVV18], the line of work I follow can be seen as the study of a formalization of a core of Theseus.

E.3.2 Reversible Pattern-Matching

In [SVV18], we propose a simple yet extensible model of reversible computation. Consider a grammar of typed patterns defined as follows:

$$\begin{array}{rcl} v, w & ::= & x \mid \star \mid \langle v, w\rangle \mid \mathtt{inl}(v) \mid \mathtt{inr}(v), \\ a, b & ::= & 1 \mid a \otimes b \mid a \oplus b, \end{array}$$

where x ranges over a set of (typed) variables, $\star$ is a unit term of unit type 1, $\langle v, w\rangle : a \otimes b$ is a pair of a value $v : a$ and a value $w : b$, and provided that $v : a$, we have $\mathtt{inl}(v) : a \oplus b$ and $\mathtt{inr}(v) : b \oplus a$. A pattern v is valid if each variable appears at most once in it. We denote with $\mathrm{FV}(v)$ the set of variables appearing in v. A closed pattern is called a *value*.

In a functional setting, a reversible computation between type a and type b is a bijection between values of the respective types. Aiming at an applicative language on structured types, in [SVV18] we follow the standard strategy [Lan66, Bur69, Tur79] and propose a syntax of isomorphisms based on pattern-matching.

An *iso* consists of a set of clauses

$$\left\{ \begin{array}{ccc} v_1 & \leftrightarrow & v_1' \\ & \vdots & \\ v_n & \leftrightarrow & v_n' \end{array} \right\},$$

where for each i, we have $\mathrm{FV}(v_i) = \mathrm{FV}(v_i')$. The iso is well-type of type $a \leftrightarrow b$ whenever each v_i is of type a and each v_j' is of type b. For the iso to be well-defined, the patterns on the left of each clause should be *non-overlapping*. For the iso to be injective, the patterns should furthermore be *exhaustive*: all values of type a should have a matching pattern in the iso. To enforce bijectivity, the same two constraints should also be set on the *right*-hand-side of the clauses of the iso.

In other word, an iso defines a bijection between values of type a and values of type b if and only if the patterns are exhaustive and non-overlapping on the left *and* on the right. For instance, the (bijective) iso ω of type $a \otimes (b \oplus c) \leftrightarrow (a \otimes b) \oplus (a \otimes c)$ can be defined

$$\left\{ \begin{array}{ccc} \langle x, \mathtt{inl}\, y\rangle & \leftrightarrow & \mathtt{inl}\, \langle x, y\rangle \\ \langle x, \mathtt{inr}\, y\rangle & \leftrightarrow & \mathtt{inr}\, \langle x, y\rangle \end{array} \right\}.$$

One can check that the left hand-side of the iso satisfies the two required properties: Any value of type $a \otimes (b \oplus c)$ is either of the form $\langle x, \mathtt{inl}\, y\rangle$ or of the form $\langle x, \mathtt{inr}\, y\rangle$.

E.3.3 A Categorical Interpretation

> Together with Kostia Chardonnet and Louis Lemonnier, we studied the categorical structure underlying the pattern-matching presented in Section E.3.2. This work yielded a publication [CLV21] presented in this section. The text is taken from the introduction of the paper.

The categorical analysis of partial injective maps have been thoroughly analyzed since 1979, first by Kastl [Kas79], and then by Cockett and Lack [CL02, CL03, CL07]. This led to the development of inverse category: a category equipped with an inverse operator in which all morphisms have partial inverses and are therefore reversible. The main aspect of this line of research is that partiality can have a purely algebraic description: one can introduce a restriction operator on morphisms, associating to a morphism a partial identity on its domain. This categorical framework has recently been put to use to develop the semantics of specific reversible programming constructs and concrete reversible languages: analysis of recursion in the context of reversibility [AK16, Kaa19, KV19], formalization of reversible flowchart languages [12, 22], analysis of side-effects [HK15, HKK18], etc. Interestingly enough however, the adequacy of the developed categorical constructs with reversible functional programming languages has been seldom studied. For instance, if Kaarsgaard et al. [KAG17] mention Theseus as a potential use-case, they do not discuss it in details. So far, the semantics of functional and applicative reversible languages has always been done in concrete categories of partial isomorphisms [KV19, KR21].

In particular, one important aspect that has not been addressed yet in detail is the categorical interpretation of pattern-matching. If pattern-matching can be added to reversible imperative languages [GKY19], it is particularly relevant in the context of functional languages where it is one of the core construct needed for manipulating structured data. This is for instance emphasized by the several existing languages making use of it [YAG12, TA15, JS14, JKT18, SVV18, CSV20, CSV23]. In the literature, pattern-matching has either been considered in the context of a Set-based semantics [GKY19], or more generally in categorical models making heavy use of rig structures [CS16] or co-products [KR21, KV19] to represent it. If such rich structures are clearly enough to capture pattern-matching, we show in [CLV21] that they are too coarse, and that a weaker structure is enough for characterizing pattern-matching.

E.3.4 Inductive Types, Fixpoints and termination

As it stands, the language is very limited: it is for instance not possible to express natural numbers, or lists. One simple solution consists in extending the language with *inductive types* and *recursion*.

Through the Curry-Howard isomorphism, inductive types are the twin siblings of inductively defined predicates. Arguably, the formalization of induction takes its root on one hand [BCMS89] from the theory of inductive definitions [Fef70, BFPS81, Acz77] and Martin Löf's type theory [Mar71, Mar84], and on the other hand from de Bakker and Scott's μ-calculus [SB69, Bak71, Pra81, Koz83]. The main design choice consists in whether to use an equational presentation with named constructors [CP90, Dyb91, Dyb94] or an *anonymous* presentation using μ-abstractions [BR72, Roe74, Men88, Mat98], in the same way λ-abstractions can be used to express (unnamed) functionals.

If in [SVV18] we use named constructors for the dedicated type constructor $[a]$ for lists, in later works [CSV20, CSV23] we work with the more generic anonymous inductive types, using a μ-construction, as follows.

$$a, b \quad ::= \quad X \mid 1 \mid a \otimes b \mid a \oplus b \mid \mu X.a.$$

The additional type constructor $\mu X.a$ comes with a pattern/value constructor $\mathtt{fold}$. The typing rules enforces the equivalence

$$\mathtt{fold}\, v : \mu X.a \quad \sim \quad v : a[X \mapsto \mu X.a],$$

encapsulating de Bakker and Scott's induction principle [SB69]. A type constructor $[a]$ for lists can for instance be defined as $[a] \triangleq \mu X.1 \oplus (a \otimes X)$. The list constructor $\mathtt{nil}$ and $v_1 : v_2$, standing respectively for the empty list and the cons operation, can be defined as $\mathtt{nil} \triangleq \mathtt{fold}\,\mathtt{inl}\,\star$ and $v_1 : v_2 \triangleq \mathtt{fold}\,\mathtt{inr}\,\langle v1, v2\rangle$.

An inductive type might have arbitrarily large values: in a functional setting, the traditional method [Jon87] consists in using *fixpoints*. To do so, one missing feature of the iso language is the capability to manipulate variables representing isos. In [SVV18] we extend the language by adding iso variables, application, lambda-abstraction over iso-variables, and a fixpoint operator $\mathtt{fix}$. Assuming $\mathtt{fix}\, f.\omega$ has the behavior

$$\mathtt{fix}\, f.\omega \to \omega[f \mapsto \mathtt{fix}\, f.\omega]$$

and assuming a suitable syntax extension for the right-hand-side of isos, we can then define the (higher-order) map operation of type $(a \leftrightarrow b) \Rightarrow ([a] \leftrightarrow [b])$ as follows:

$$\mathtt{map} \quad \triangleq \quad \lambda g.\mathtt{fix}\, f. \left\{ \begin{array}{lcl} \mathtt{nil} & \leftrightarrow & \mathtt{nil} \\ h : t & \leftrightarrow & (g\, h) : (f\, h) \end{array} \right\}.$$

Assuming the iso g is of type $a \leftrightarrow b$, the operation $\mathtt{map}\, g$ sends $\mathtt{nil}$ to $\mathtt{nil}$ and otherwise applies g to the head of the list and itself to the tail.

Introducing fixpoints pose the question of the termination of programs: without termination, the isos describes injective maps between sets of values. Bijections are only obtained in the case of *terminating* isos. One of the result of [SVV18] is to formalize this fact.

E.3.5 Pattern-Matching for Quantum Control

The reversible iso-language described in Section E.3.4 is amenable to the same algebraic extension as the vectorial lambda calculus presented in Section E.1.3.3: values are extended to *linear combinations*. Therefore, assuming $\mathtt{tt} \triangleq \mathtt{inl}\star$ and $\mathtt{ff} \triangleq \mathtt{inr}\star$ are the two (standard) values of type $1 \oplus 1$, one can now consider values of the form $\alpha \cdot \mathtt{tt} + \beta \cdot \mathtt{ff}$, where α, β are scalars: the type $1 \oplus 1$ is a 2-dimensional module. In [SVV18] we formalize an algebraic extension of the iso-language supporting the encoding of *unitary* maps (when scalars ranges over complex numbers). For instance, the Hadamard gate can be represented with the iso

$$\left\{ \begin{array}{lcl} \mathtt{ff} & \leftrightarrow & \frac{\sqrt{2}}{2}(\mathtt{ff} + \mathtt{tt}) \\ \mathtt{tt} & \leftrightarrow & \frac{\sqrt{2}}{2}(\mathtt{ff} - \mathtt{tt}) \end{array} \right\}$$

of type $(1 \oplus 1) \leftrightarrow (1 \oplus 1)$. In the paper, we discuss how this idea can be extended to types of infinite dimension such as lists, and we provide a (restricted) formal setting for the support of lists and linear combinations. In particular, we provide a compositional interpretation of isos as unitaries in ℓ^2-spaces.

The quantum SWITCH. In the context of the algebraic extension of the iso-language, the quantum SWITCH is simple to write. Indeed, suppose that $u : a \leftrightarrow b$ and $v : b \leftrightarrow a$ are two isos, then one can write the iso

$$\left\{ \begin{array}{lcl} \mathtt{inl}\, x & \leftrightarrow & \mathtt{inl}(v\,(u\, x)) \\ \mathtt{inr}\, x & \leftrightarrow & \mathtt{inr}(u\,(v\, x)) \end{array} \right\}$$

of type $(a \oplus b) \leftrightarrow (a \oplus b)$. Depending on the branch (left or right), either $u\, v$ or $v\, u$ is applied. The remaining question is whether u and v are indeed "used" only once.

E.3.6 Relationship with the Logic μMALL

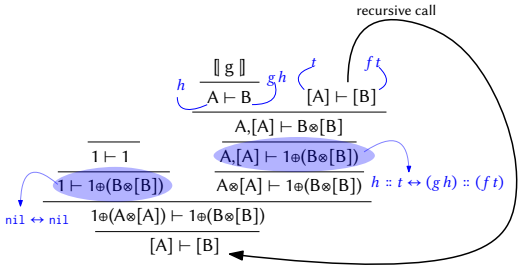
> This section summarizes a series of papers obtained with Kostia Chardonnet, a Ph.D student I co-supervised with Alexis Saurin [CSV20, CSV21, CSV23, Cha23].

The extended type system presented in Section E.3.4 is very reminiscent of the logic µMALL, an extension of MALL with inductive formulae. Introduced by Baelde, [BM07, Bae08] the logic µMALL is linear and features tensors and coproducts. Inductive formulae are dealt with infinite proof structures. The problem in this context is then to characterize whether a given infinite derivation is indeed correct: there exist several validity criteria [Bae12, BDS16, Dou17, NST18]. The strict linearity of the logic together its ability to express inductive formulae makes µMALL a good candidate logic to serve as a Curry-Howard correspondence with the iso-language sketched in Section E.3.4.

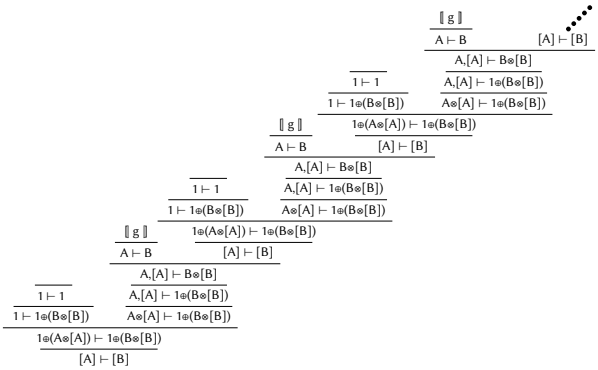
As mentioned earlier, an iso $\omega : A \leftrightarrow B$ corresponds to both a computation sending a value of type A to a result of type B and a computation sending a value of type B to a result of type A. The logical counterpart consists in two proofs: a proof π of $A \vdash B$ and a proof $\pi^{\perp}$ of $B \vdash A$. These proofs describes an isomorphism if the two possible cuts between them reduce to an identity proof.

The case of *structurally recursive* isos is considered in [CSV23]. We show that the resulting language can encode any primitive recursive function [Har87], and we give an interpretation of well-typed isos as proofs of isomorphisms in µMALL. The work described in [CSV23] discusses how the syntactical constraints of structural recursion are linked to the validity of infinite proofs corresponding to programs.

The example discussed in the paper is the proof $[\![\mathtt{map}\, g]\!]$ corresponding to the term $\mathtt{map}\, g$ with $g : A \leftrightarrow B$ being



and where $[\![g]\!]$ is the proof corresponding to g. Although the proof is folded in a cyclic manner, it corresponds to an infinite proof: the back-edge is really an infinite branch consisting of "copy-and-paste" of the proof structure:



The fact that proof is valid comes from the infinite number of left-unfolding of the list.

[Val10b] Benoît Valiron. "Semantics of a typed algebraic lambda-calculus". In: *Proceedings of the Sixth Workshop on Developments in Computational Models: Causality, Computation, and Physics, DCM 2010* (Edinburgh, Scotland, July 9–10, 2010). Ed. by S. Barry Cooper, Prakash Panangaden, and Elham Kashefi. Vol. 26. Electronic Proceedings in Theoretical Computer Science. Preliminary work to the journal paper [Val13a]. 2010, pp. 147–158. DOI: 10.4204/EPTCS.26.14.

[ADV11] Pablo Arrighi, Alejandro Díaz-Caro, and Benoît Valiron. "Subject reduction in a curry-style polymorphic type system with a vectorial structure". In: *Proceedings of the 7th International Workshop on Developments of Computational Methods, DCM 2011* (Zurich, Switzerland, July 3, 2021). Ed. by Elham Kashefi, Jean Krivine, and Femke van Raamsdonk. Vol. 88. Electronic Proceedings in Theoretical Computer Science. Preliminary work to the journal paper [ADV17]. 2011, pp. 1–15. DOI: 10.4204/EPTCS.88.1. HAL: hal-00924926. ARXIV: 1012.4032.

[CDPV13] G. Chiribella, G. M. D'Ariano, P. Perinotti, and B. Valiron. "Quantum computations without definite causal structure". In: *Physical Review A* 88 (2013), p. 022318. DOI: 10.1103/PhysRevA.88.022318. ARXIV: 0912.0195.

[Val13a] Benoît Valiron. "A typed, algebraic, computational lambda-calculus". In: *Mathematical Structures in Computer Science* 23.2 (2013). Journal, extended version of [Val10b]., pp. 504–554. DOI: 10.1017/S0960129512000205.

[ADPTV14] Ali Assaf, Alejandro Díaz-Caro, Simon Perdrix, Christine Tasson, and Benoît Valiron. "Call-by-value, call-by-name and the vectorial behaviour of the algebraic lambda-calculus". In: *Logical Methods in Computer Science* 10.4 (2014). DOI: 10.2168/LMCS-10(4:8)2014. ARXIV: 1005.2897v7.

[VZ14a] Benoît Valiron and Steve Zdancewic. "Finite vector spaces as model of simply-typed lambda-calculi". In: *Proceedings of the 11th International Colloquium on Theoretical Aspects of Computing, ICTAC 2014* (Bucharest, Romania, Sept. 17–19, 2014). Ed. by Gabriel Ciobanu and Dominique Méry. Vol. 8687. Lecture Notes in Computer Science. See [VZ14b] for the long version. Springer, 2014, pp. 442–459. DOI: 10.1007/978-3-319-10882-7_26.

[VZ14b] Benoît Valiron and Steve Zdancewic. "Modeling simply-typed lambda calculi in the category of finite vector spaces". In: *Scientific Annals of Computer Science* 24.2 (2014), pp. 325–368. DOI: 10.7561/SACS.2014.2.325.

[ADV17] Pablo Arrighi, Alejandro Díaz-Caro, and Benoît Valiron. "The vectorial lambda-calculus". In: *Information and Computation* 254 (2017), pp. 105–139. DOI: 10.1016/j.ic.2017.04.001. HAL: hal-00921087.

[SVV18] Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. "From symmetric pattern-matching to quantum control". In: *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2018* (Thessaloniki, Greece). Ed. by Christel Baier and Ugo Dal Lago. Vol. 10803. Lecture Notes in Computer Science. Springer, 2018, pp. 348–364. DOI: 10.1007/978-3-319-89366-2_19. HAL: hal-01763568. ARXIV: 1804.00952.

[DGMV19] Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel, and Benoît Valiron. "Realizability in the unitary sphere". In: [LICS19], pp. 1–13. DOI: 10.1109/LICS.2019.8785834. HAL: hal-02175168. ARXIV: 1904.08785.

[CSV20] Kostia Chardonnet, Alexis Saurin, and Benoît Valiron. "Toward a Curry-Howard equivalence for linear, reversible computation - work-in-progress". In: [RC20], pp. 144–152. DOI: 10.1007/978-3-030-52482-1_8. HAL: hal-03103455.

[CLV21] Kostia Chardonnet, Louis Lemonnier, and Benoît Valiron. "Categorical semantics of reversible pattern-matching". In: [MFPS21], pp. 18–33. DOI: 10.4204/EPTCS.351.2.

[CSV23] Kostia Chardonnet, Alexis Saurin, and Benoît Valiron. "Towards a Curry-Howard correspondence for linear, reversible computation". In: *Proceedings of the 5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)* (Rome (virtual), Italy). 2021. HAL: lirmm-03271484.

Table E.1: Personal publications related to Chapter E.

Chapter F

Opening

Over the past dozen years, the field of quantum programming languages has experienced substantial maturation. Fifteen years ago, programming languages were restricted to toy languages, independent from hardware, and only relied on small mathematical specifications such as Knill's QRAM model [Kni96]. What happened below this layer was left "to the physicists". Nowadays, quantum programming languages address large, industrial-scale problem instances and target concrete hardware [McKinsey21]. It has become clear that many low-level aspects require scrutiny and are of interest to the programmer: Timing, coprocessor expressivity power, quantum superposition of execution, etc. All of this needs to be taken into account at once, and it is not clear whether this can be compartmented into an intermediate representation, with the compiler responsible for the translation, or if some or all of the constraints have to be lifted to the high-level language. Both points of view are interesting: leveraging intermediate representations for quantum computation to this extended paradigm but also having a full-fledged high-level language for describing such processes.

Our seminal work on Quipper is arguably a milestone in the design of quantum programming languages [GLRSV13b]. Quipper can indeed be seen as an experiment in the design of a scalable language with sound principles. The central axiom is that programming quantum algorithms is, first of all, the description of the construction of a circuit: any quantum programming language should, therefore, be a classical programming language providing a series of specialized constructs to realize circuits as efficiently as possible. Several design principles ensue. For instance, one such design is the capability to box functions—i.e., considering a function from qubit to qubit as a circuit— and unbox circuits—that is, considering a circuit as a function acting on a qubit. Another principle is the ability to build and apply higher-order circuit combinators: the programming language should make it easy to control a circuit, use it in the context of local, ancilla wires, and otherwise perform arbitrary transformations on it. A last principle worth mentioning is the ability to generate a circuit from a classical description. These principles have been laid out in the context of Quipper: they are still state-of-the-art in the design of current quantum programming languages.

The field is, however, moving fast, and as NISQ [Pre18] reaches the level of industrialization, new paradigms are needed. On the one hand, the so-called realm of *hybrid computation* requires more than circuit-description languages: a distributed programming model where the classical processor and the quantum coprocessor speak on equal grounds. Programming in this context opens novel questions. For instance, what is the classical expressive power of the quantum coprocessor compared to that of the classical processor? What are the programming features needed within the quantum coprocessor, i.e., what are the reasonable capabilities of the coprocessor in terms of memory, clock, and timing related in the interactions? The large-scale development of quantum hardware also opens the door to the question of quantum compilation. Quantum circuits can no longer be considered low-level targets, and a

quantum programming environment cannot be reduced to circuit description and evaluation [Qui20, BISG+20, HFGB+23, Qis, Mei24].

From a theoretical standpoint, in the last 15 years, we have proposed a novel computational construct: quantum SWITCH [CDPV13]. This construct can be summarized by asking whether it is conceivable to realize *in superposition* the sequence "U then V" and the sequence "V then U", given only one single copy of each. Said otherwise, instead of only having data in superposition, is it possible to also have executions in superposition? If U and V are unitary, the overall operation is unitary, so this should not be a problem. The issue's crux is the impossibility of building a circuit with only two holes–one for U and one for V–when realizing this procedure.

Although this "quantum test" makes sense from a computational perspective and is mathematically meaningful, it lies outside the circuit model. Nonetheless, it has been realized by concrete physical experiment [PMAC+15, TCMG+21]. This exemplifies the cross-fertilization happening between physics and computer science: The scientific community has derived several exciting results from the realization of quantum superposition of executions, including speedup in communication and finer analysis of quantum metrology [ACB14, TCMG+21, ZYC20]. From the quantum programming language side, the promise of quantum control lies in the *programming* of quantum unitaries. Instead of constructing circuits as lists of opaque low-level gates, quantum control opens the door to a complete programming environment manipulating classical and quantum operations within a unified paradigm.

Since 2008, in parallel with the development of quantum programming languages, semantics have seen a significant development [RS18a, LMZ18, Wes19]. Indeed, semantics is essential to unearth the structures underlying quantum computation and to shed light on the suitable structures for manipulating quantum computation soundly and consistently. The development of semantics for a programming language is, in particular, what makes it possible to obtain language-supporting techniques to express and prove the properties of programs.

At the dawn of 2010, the semantics of quantum programming languages were either very abstract or very close to physics textbooks: based on superoperators and completely positive maps, following the standard models of quantum information theory, or, on the opposite, following purely categorical constructions [SV08a, SV08b, Mal10]. Since then, the models have tremendously evolved, capturing more fine-grained concrete, useful programmatic paradigms. An example is the interplay between quantum circuits and measurements in the context of circuit-description languages: we now have mathematical representations of dynamic circuits, where the circuit's shape might depend on the result of previous measurements. Another example is the range of sophisticated type systems based on linear logic. Such type systems can now enforce the non-duplicability of quantum data and, at the same time, provide refined logical properties by relying on dependent types [FKS20]. Finally, a last example concerns quantum control and superposition of executions: we now have several proposals of formal languages equipped with rewrite systems able to formalize what it means to feature quantum superposition of executions.

A Few Current Trends of Research

The rest of the chapter broadens the focus to a set of ongoing trends of research.

Rise of Graphical Languages. In the field of quantum programming languages, one crucial event in recent years has been the advent of graphical languages targeted toward quantum computation. Arguably, the first one is the ZX calculus . Abstracting away from quantum circuits, the language is a formal graph-based language akin to what is used in the context of tensor networks but specialized for the specificities of quantum computation. In ZX, wires correspond to qubit states, and the available nodes in the graph stand for linear maps related to elementary operations such as rotations along X and Z basis. Unlike quantum circuits,

it is equipped with sound semantics and a complete equational theory, making it a strong candidate for reasoning and processing quantum computation.

Over time, the ZX calculus has proven more versatile than quantum circuits when considering several classes of problems such as circuit optimization, qubit layout, post-selection, or reasoning over error-correcting schemes [pub22]. The language is also easily extensible. It has, for instance, been extended to support features such as measurement and trace, or arrays of qubits. Compared to syntactic, more conventional approaches such as QASM implementing plain quantum circuits, the ZX calculus has shown to be a credible approach for serving as an intermediate representation in the context of a quantum compilation toolchain [Mei24].

The success of the ZX calculus has spurred a line of research in the design of graphical languages with a focus on specific backends. Indeed, the ZX calculus is particularly well-suited for hardware based on the gate-set Clifford+T but possibly less fitted for other models. For instance, for Clifford+Toffoli—the canonical gate-set for cat-qubits—the ZH calculus is better suited [Vil18]. Similarly, the ZW calculus [HNW18] is considered adequate for reasoning on quantum computation with Rydberg atoms. Recently, one can also cite languages such as the LOv-calculus developed for linear optical circuits [CHMPV22].

These trends emphasize connecting theoretical computer science concepts with physical implementations. Such a cross-disciplinary integration has already been shown to be effective: we have recently shown a completeness result for quantum circuits based on the development of the LOv-calculus [CHMPV23a].

The story is still ongoing, and exciting questions await us. For instance, existing languages currently only model quantum computation in finite-dimensional spaces; the ability of graphical languages to handle infinite-dimensional objects is still work in progress [FC22, SYG24]. Another question is the interoperability of these languages. In particular, if the Kronecker product is the canonical operation to join systems together in ZX-based languages, linear-optical languages use the product. Finally—and maybe more generally—these languages are still far from many considerations closer to the hardware: they do not handle (yet) timing, noise, nor hybrid computation.

Unification of Quantum and Classical Control. Hybrid quantum computation is a model of computation where the interaction with the quantum coprocessor can depend on the results of intermediate measurements. In the standard model, the coprocessor is considered a closed box that can be manipulated using an interface given once for all. In particular, the elementary operations available in the quantum coprocessor are not programmable.

The model of quantum control instead considers the case where the programmer can describe a superposition of executions instead of a simple list of gates. In this model, one can express the purely quantum part with native, quantum-specific programming constructs such as the quantum SWITCH [CDPV13].

One missing aspect of quantum control is the interaction with the classical machine. Indeed, current descriptions of quantum control only focus on the purely quantum part, and the model does not encompass a hybrid system where, for instance, quantum information could be measured, let alone used to drive quantum evolution. Therefore, finding a model unifying both classical and quantum control in the same framework is an open question.

Another more foundational issue is how to handle quantum, recursive datatypes. Consider, for instance, the type of lists of qubits and the element consisting of a list of size 2 and a list of size 5. What does it mean to iterate over this superposition of lists? The question can be generalized: What kind of recursion is allowed in a purely quantum context? This question lies in the more general problem of the expressive power of quantum control. Indeed, if superpositions of programs makes a powerful computational paradigm, the constraint of unitarity is a subtle condition to enforce at the syntactic level.

A last large research avenue open for quantum control is the problem of turning the description of a superposition of executions into something that can be physically executed on a quantum backend. The question is twofold. On the one hand, several non-standard models

of quantum computation feature some notion of quantum control, such as AQG or routed circuits. Compiling on these (formal) backends is already a stimulating question. On the other hand, we do have concrete hardware candidates for (physical) quantum computation. A natural question is to study how much quantum control these hardware candidates can handle and devise a suitable compilation scheme for them.

Quantum Compilation Toolchain. Although the field of quantum programming languages is now reaching a mature state, turning a quantum program into a realistic set of low-level operations executable on a quantum coprocessor is still a work in progress. Currently, each vendor offers a specific solution, usually with a particular quantum dialect and a compilation framework specifically tailored for one particular hardware. Existing quantum compilers are also currently very limited. For once, they provide little parametrization and do not scale well: they mainly target (small) NISQ devices, and LSQ is still an open field of research. Another issue is the ability to handle hybrid computation; static circuits remain the norm of what is possible to compile.

A crucial open question in this realm consists in devising tools and techniques to effectively compile quantum programs down to low-level, executable physical operation. Following what has been done in the classical setting, it would be natural to consider one or several intermediate representations specific to quantum computation, such as graphical languages. In any case, problems such as timing and parallelism of quantum operations and topological constraints of the hardware (or of the quantum error-correcting layer) need to be addressed in a consistent manner. Because of the very distinct kinds of hardware, it is now admitted that there will not be a one-fit-all solution. Nonetheless, many problems are cross-platform, yielding similar answers. Although a generic hardware-independent rigid compiler might not be doable, devising a common framework for building and composing compiling tools and modules is an active research area.

A complementary problem in quantum compilation concerns optimizing and estimating the resources needed to run a given quantum program. Although this program is written in a hardware-agnostic, higher-order language, the target backend is one specific coprocessor. The problem is akin to what happens for critical systems: This processor has limited memory. Error correction—if any—is costly: we want the code to be as optimized and as parallel as possible. This tension calls for developing optimization schemes to minimize resources and static analysis tools to evaluate these resources.

Static Analysis for Quantum Programs. Verification techniques for quantum programs are currently in their early stages of development. So far, various attempts have been followed with little of a unified approach [CBLVVX21]. Among these, one can mention the use of proof assistants such as Coq and Isabelle/HOL to prove properties of quantum programs, but also deductive verification techniques, either standalone and based on a crafted Hoare logic or embedded in existing tools such as Why3. Although other novel SMT-based tools have recently emerged, the field still needs to be more structured, making it challenging to compare these methods and ascertain the overall direction of the discipline.

The main problem with static analysis of quantum programs is the nature of the thing we want to analyze. Many aspects can be considered. If the program describes a static circuit, as discussed in the trend related to quantum compilation, one can be interested in the circuit's size, shape, or depth. Another critical aspect is ensuring that the circuit generated by the program implements the correct unitary map. In the context of a probabilistic algorithm relying on measurements, one should also guarantee the probability of success.

In the context of a quantum compilation toolchain, one can also be interested in certifying other layers of the compilation stack, such as the optimization schemes, the qubit layout process, and, in general, circuit transformations and translations to dedicated graphical, intermediate representations.

Finally, an untouched aspect of the static analysis of quantum programs is quantum con-

trol. What can be asserted in this context, how to do it, and how to verify it is still a completely open area of research.

The questions discussed above rely on sound, expressive semantics of both quantum programs and layers in the quantum compilation toolchain. Unlike classical computation, where models are discrete structures, in quantum computation, the mix of discrete and continuous structure, duplicable and non-duplicable objects, linear algebra, and operator theory renders the development of powerful analysis tools challenging without a fine-grain understanding of the structures at stake. The development of semantics for quantum programming languages is a continuous dialog between three actors: physics, hinting at the underlying constraints; computer science, discovering unknown computational structures and techniques; and semantics, formalizing them and providing a firm, sound framework upon which to build and reason.

Conclusion

This thesis has presented how I understand the evolution of quantum programming language within the past fifteen years. In this time frame, the field of quantum programming languages has shifted from toy examples to a more mature state. A dialog between physics and computer science has fostered unforeseen discoveries along this path. The story is, however, not over, and the field remains open, presenting exciting questions and opportunities for future developments.
I am already involved in some of the research paths presented in this section, in particular with currently ongoing Ph.D students. With Nicolas Heurtel, Ph.D funded by a CIFRE with Quandela, we are studying graphical languages for quantum linear optics—this yielded for instance the LOv calculus [CHMPV22]. With Julien Lamiroy, co-supervised with Renaud Vilmart, we are investigating graphical languages for quantum control. With Jérome Ricciardi, Ph.D funded by CEA and co-supervised with Christophe Chareton, we are studying static analysis methods for quantum programs with measurements.

[QPL20b] Benoît Valiron, Shane Mansfield, Pablo Arrighi, and Prakash Panangaden, eds. *Proceedings 17th International Conference on Quantum Physics and Logic, QPL 2020* (Online (due to Covid), June 2–6, 2020). Vol. 340. EPTCS. 2020.

[CVV21] Kostia Chardonnet, Benoît Valiron, and Renaud Vilmart. "Geometry of interaction for ZX diagrams". In: *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021* (Tallinn, Estonia). Ed. by Filippo Bonchi and Simon J. Puglisi. Vol. 202. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2021, 30:1–30:16. ISBN: 978-3-95977-201-3. DOI: 10.4230/LIPIcs.MFCS.2021.30.

[BBVMA21] Timothee Goubault De Brugiere, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Reducing the depth of linear reversible quantum circuits". In: *IEEE Transactions on Quantum Engineering* 2 (2021), p. 3102422. DOI: 10.1109/TQE.2021.3091648. HAL: hal-03553916.

[GBVMA21] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Gaussian elimination versus greedy methods for the synthesis of linear reversible circuits". In: *ACM Transactions on Quantum Computing* 2.3 (2021), p. 11. DOI: 10.1145/3474226. HAL: hal-03547117.

[Val22] Benoît Valiron. "Semantics of quantum programming languages: classical control, quantum control". In: *Journal of Logical and Algebraic Methods in Programming* 128 (2022), p. 100790. DOI: 10.1016/J.JLAMP.2022.100790. HAL: hal-04038653.

[BBVMA22] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Decoding techniques applied to the compilation of CNOT circuits for NISQ architectures". In: *Science of Computer Programming* 214 (2022), p. 102726. DOI: 10.1016/J.SCICO.2021.102726. HAL: hal-03547113.

[CHMPV22] Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. "LOv-calculus: a graphical language for linear optical quantum circuits". In: *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*. Ed. by Stefan Szeider, Robert Ganian, and Alexandra Silva. Vol. 241. LIPIcs. 2022, 35:1–35:16. DOI: 10.4230/LIPICS.MFCS.2022.35. URL: https://doi.org/10.4230/LIPIcs.MFCS.2022.35.

[CVVV22] Kostia Chardonnet, Marc de Visme, Benoît Valiron, and Renaud Vilmart. "The Many-Worlds Calculus: Representing Quantum Control". 2022.

[CTV23] Théodore Chapuis-Chkaiban, Zeno Toffano, and Benoît Valiron. "On new pagerank computation methods using quantum computing". In: *Quantum Information Processing* 22.3 (2023), p. 138. DOI: 10.1007/S11128-023-03856-Y. HAL: hal-04056045.

[ACCRV23] Pablo Arrighi, Christopher Cedzich, Marin Costes, Ulysse Rémond, and Benoît Valiron. "Addressable quantum gates". In: *ACM Transactions on Quantum Computing* 4.3 (2023), pp. 1–41. DOI: 10.1145/3581760. HAL: hal-03936367. ARXIV: 2109.08050.

[HFGB+23] Nicolas Heurtel, Andreas Fyrillas, Grégoire de Gliniasty, Raphaël Le Bihan, Sébastien Malherbe, Marceau Pailhas, Eric Bertasi, Boris Bourdoncle, Pierre-Emmanuel Emeriau, Rawad Mezher, Luka Music, Nadia Belabas, Benoît Valiron, Pascale Senellart, Shane Mansfield, and Jean Senellart. "Perceval: a software platform for discrete variable photonic quantum computing". In: *Quantum* 7 (2023), p. 931. DOI: 10.22331/Q-2023-02-21-931. HAL: hal-03874624.

[HMSV23] Nicolas Heurtel, Shane Mansfield, Jean Senellart, and Benoît Valiron. "Strong simulation of linear optical processes". In: *Computer Physics Communications* 291 (2023), p. 108848. DOI: 10.1016/J.CPC.2023.108848. HAL: hal-03874624v1.

[CHMPV23a] Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. "A complete equational theory for quantum circuits". In: *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*. IEEE, 2023, pp. 1–13. DOI: 10.1109/LICS56636.2023.10175801. HAL: hal-03926757.

[CHMPV23b] Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. "A Complete Equational Theory for Quantum Circuits". Presentation accepted at the 18th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2023), in Aveiro, Portugal. 2023. HAL: hal-04318291v1.

[QPL23b] Shane Mansfield, Benoît Valiron, and Vladimir Zamdzhiev, eds. *Proceedings of the Twentieth International Conference on Quantum Physics and Logic, QPL 2023* (Paris, France, July 17–21, 2023). Vol. 384. EPTCS. 2023. DOI: 10.4204/EPTCS.384.

Table F.1: Personal publications since ~2020 not yet mentioned.

Bibliography

[ABGV18] C. Allouche, M. Baboulin, T. Goubault de Brugière, and B. Valiron. "Reuse method for quantum circuit synthesis". In: *Recent Advances in Mathematical and Statistical Methods, post-proceedings of the IV AMMCS International Conference on Applied Mathematics, Modeling and Computational Science, Waterloo, Canada, August 20 – 25, 2017*. Ed. by D. Marc Kilgour, Herb Kunze, Roman Makarov, Roderick Melnik, and Xu Wang. Springer International Publishing, 2018, pp. 3–12. ISBN: 978-3-319-99719-3. DOI: 10.1007/978-3-319-99719-3_1. HAL: hal-01711378.

[ABIMBK19] Juan Miguel Arrazola, Thomas R Bromley, Josh Izaac, Casey R Myers, Kamil Brádler, and Nathan Killoran. "Machine learning method for state preparation and gate synthesis on photonic quantum computers". In: *Quantum Science and Technology* 4.2 (Jan. 2019), p. 024004. DOI: 10.1088/2058-9565/aaf59e.

[Abr05] Samson Abramsky. "A structural approach to reversible computation". In: *Theoretical Computer Science* 247.3 (2005), pp. 441–464. DOI: 10.1016/j.tcs.2005.07.002. arXiv: 1111.7154.

[Abr07] Samson Abramsky. "Temperley-Lieb algebra: from knot theory to logic and computation via quantum mechanics". In: *Mathematics of Quantum Computation and Quantum Technology*. Ed. by Goong Chen, Louis Kauffman, and Samuel J. Lomonaco. New York: Chapman, Hall/CRC, Taylor, and Francis, 2007. Chap. 15, pp. 415–458. ISBN: 978-1-58488-900-7. DOI: 10.1201/9781584889007. arXiv: 0910.2737.

[Abr93] Samson Abramsky. "Computational interpretations of linear logic". In: *Theoretical Computer Science* 111.1-2 (1993), pp. 3–57. DOI: 10.1016/0304-3975(93)90181-R.

[AC04] Samson Abramsky and Bob Coecke. "A categorical semantics of quantum protocols". In: [LICS04], pp. 415–425. DOI: 10.1109/LICS.2004.1319636. arXiv: quant-ph/0402130.

[ACB14] Mateus Araújo, Fabio Costa, and Časlav Brukner. "Computational advantage from quantum-controlled ordering of gates". In: *Physical Review Letters* 113 (25 2014), p. 250402. DOI: 10.1103/PhysRevLett.113.250402. arXiv: 1401.8127.

[Acc15] Beniamino Accattoli. "Proof Nets and the Call-by-Value Lambda-Calculus". In: *Theoretical Computer Science* 606 (2015), pp. 2–24. DOI: 10.1016/j.tcs.2015.08.006.

[ACCRV23] Pablo Arrighi, Christopher Cedzich, Marin Costes, Ulysse Rémond, and Benoît Valiron. "Addressable quantum gates". In: *ACM Transactions on Quantum Computing* 4.3 (2023), pp. 1–41. DOI: 10.1145/3581760. HAL: hal-03936367. arXiv: 2109.08050.

[ACRŠZ10] Andris Ambainis, Andrew M. Childs, Ben Reichardt, Robert Špalek, and Shengyu Zhang. "Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer". In: *SIAM Journal on Computing* 39.6 (2010), pp. 2513–2530. DOI: 10.1137/080712167.

[Acz77] Peter Aczel. "An introduction to inductive definitions". In: *Handbook of Mathematical Logic*. Ed. by John Barwise. Vol. 90. Studies in Logic and the Foundations of Mathematics. North Holland, 1977, pp. 739–782. DOI: 10.1016/S0049-237X(08)71120-0.

[AD05] Pablo Arrighi and Gilles Dowek. "A computational definition of the notion of vectorial space". In: *Proceedings of the Fifth International Workshop on Rewriting Logic and its Applications (WRLA'04)*. Vol. 117. Electronic Notes in Theoretical Computer Science. 2005, pp. 249–261. DOI: 10.1016/j.entcs.2004.06.013.

[AD08] Pablo Arrighi and Gilles Dowek. "Linear-algebraic lambda-calculus: higher-order, encodings, and confluence." In: *Proceedings of the 19th International Conference on Rewriting Techniques and Applications, RTA'08* (Hagenberg, Austria). Ed. by Andrei Voronkov. Vol. 5117. Lecture Notes in Computer Science. Springer, 2008, pp. 17–31. ISBN: 978-3-540-70588-8. DOI: 10.1007/978-3-540-70590-1_2.

[AD09] Pablo Arrighi and Alejandro Díaz-Caro. "Scalar system F for linear-algebraic lambda-calculus: towards a quantum physical logic". In: *Proceedings of the 6th International Workshop on Quantum Physics and Logic, QPL'09* (Oxford, UK.). Ed. by B. Coecke, P. Panangaden, and P. Selinger. Vol. 270-2. Electronic Notes in Theoretical Computer Science. 2009, pp. 219–229. DOI: 10.1016/j.entcs.2011.01.033.

[AD12a] Pablo Arrighi and Alejandro Díaz-Caro. "A system F accounting for scalars". In: *Logical Methods in Computer Science* 8.1 (2012). See also extended abstract [AD09]. DOI: 10.2168/LMCS-8(1:11)2012.

[AD12b] Pablo Arrighi and Gilles Dowek. "The physical Church-Turing thesis and the principles of quantum theory". In: *International Journal of Foundations of Computer Science* 23.5 (2012), pp. 1131–1146. DOI: 10.1142/S0129054112500153. arXiv: 1102.1612.

[AD17] Pablo Arrighi and Gilles Dowek. "Lineal: a linear-algebraic lambda-calculus". In: *Logical Methods in Computer Science* 13.1 (2017). DOI: 10.23638/LMCS-13(1:8)2017.

[Ada02] Andrew Adamatzky, ed. *Collision-Based Computing*. Springer-Verlag, 2002. URL: http://www.cems.uwe.ac.uk/~aadamatz/compiled.htm.

[ADPTV14] Ali Assaf, Alejandro Díaz-Caro, Simon Perdrix, Christine Tasson, and Benoît Valiron. "Call-by-value, call-by-name and the vectorial behaviour of the algebraic lambda-calculus". In: *Logical Methods in Computer Science* 10.4 (2014). DOI: 10.2168/LMCS-10(4:8)2014. arXiv: 1005.2897v7.

[ADV11] Pablo Arrighi, Alejandro Díaz-Caro, and Benoît Valiron. "Subject reduction in a curry-style polymorphic type system with a vectorial structure". In: *Proceedings of the 7th International Workshop on Developments of Computational Methods, DCM 2011* (Zurich, Switzerland, July 3, 2021). Ed. by Elham Kashefi, Jean Krivine, and Femke van Raamsdonk. Vol. 88. Electronic Proceedings in Theoretical Computer Science. Preliminary work to the journal paper [ADV17]. 2011, pp. 1–15. DOI: 10.4204/EPTCS.88.1. HAL: hal-00924926. arXiv: 1012.4032.

[ADV17] Pablo Arrighi, Alejandro Díaz-Caro, and Benoît Valiron. "The vectorial lambda-calculus". In: *Information and Computation* 254 (2017), pp. 105–139. DOI: 10.1016/j.ic.2017.04.001. HAL: hal-00921087.

[AG05a] Thorsten Altenkirch and Jonathan Grattage. "A functional quantum programming language". In: *Proceedings of the 20th Symposium on Logic in Computer Science, LICS'05* (Chicago, Illinois, US.). Ed. by Prakash Panangaden. IEEE. IEEE Computer Society Press, 2005, pp. 249–258. DOI: 10.1109/LICS.2005.1. arXiv: quant-ph/0409065.

[AG05b] Thorsten Altenkirch and Jonathan Grattage. "QML: Quantum data and control". Draft, extended version of the LICS publication [AG05a]. 2005.

[AG09] Thorsten Altenkirch and Alexander S Green. "The quantum IO monad". In: [GM09], pp. 173–205.

[AG13] Holger Bock Axelsen and Robert Glück. "Reversible representation and manipulation of constructor terms in the heap". In: [RC13], pp. 96–109. DOI: 10.1007/978-3-642-38986-3_9.

[AGVS05] Thorsten Altenkirch, Jonathan Grattage, Juliana Kaizer Vizzotto, and Amr Sabry. "An algebra of pure quantum programming". In: [QPL07], pp. 23–47. DOI: 10.1016/j.entcs.2006.12.010.

[AK16] Holger Bock Axelsen and Robin Kaarsgaard. "Join inverse categories as models of reversible recursion". In: *Proceedings of the 19th International Conference on Foundations of Software Science and Computation Structures, FoSSaCS'16* (Eindhoven, The Netherlands). Ed. by Bart Jacobs and Christof Löding. Vol. 9634. Lecture Notes in Computer Science. Springer, 2016, pp. 73–90. DOI: 10.1007/978-3-662-49630-5_5.

[AM17] Pablo Arrighi and Simon Martiel. "Quantum causal graph dynamics". In: *Physical Review D* 96 (2 2017), p. 024026. DOI: 10.1103/PhysRevD.96.024026. arXiv: 1607.06700.

[AM97] Samson Abramsky and Guy McCusker. "Call-by-value games". In: *Computer Science Logic, 11th International Workshop, CSL'97 Annual Conference of the EACSL* (Aarhus, Denmark). Ed. by Mogens Nielsen and Wolfgang Thomas. Vol. 1414. Lecture Notes in Computer Science. European Association for Computer Science Logic. Springer Verlag, Aug. 1997, pp. 1–17. DOI: 10.1007/BFb0028004.

[Amb12] Andris Ambainis. "Variable time amplitude amplification and quantum algorithms for linear algebra problems". In: *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012* (Paris, France, Feb. 29–Mar. 3, 2012). Ed. by Christoph Dürr and Thomas Wilke. Vol. 14. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 636–647. ISBN: 978-3-939897-35-4. DOI: 10.4230/LIPIcs.STACS.2012.636.

[AMM14] Matthew Amy, Dmitri Maslov, and Michele Mosca. "Polynomial-time T-depth optimization of Clifford+t circuits via matroid partitioning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.10 (2014), pp. 1476–1489. DOI: 10.1109/TCAD.2014.2341953.

[AMMR13] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.6 (2013), pp. 818–830. DOI: 10.1109/TCAD.2013.2244643.

[Amy13] Matthew Amy. "Algorithms for the Optimizations of Quantum Circuits". MA thesis. University of Waterloo, 2013.

[Amy18] Matthew Amy. "Towards large-scale functional verification of universal quantum circuits". In: [QPL19], pp. 1–21. DOI: 10.4204/EPTCS.287.1.

[Amy19] Matthew Amy. "Formal Methods in Quantum Circuit Design". PhD thesis. University of Waterloo, Ontario, Canada, 2019. URL: http://hdl.handle.net/10012/14480.

[AR02] Andrea Asperti and Luca Roversi. "Intuitionistic light affine logic". In: *ACM Transactions on Computational Logic* 3.1 (2002), pp. 137–175. DOI: 10.1145/504077.504081.

[Arr19] Pablo Arrighi. "An overview of quantum cellular automata". In: *Natural Computing* 18.4 (2019), pp. 885–899. DOI: 10.1007/s11047-019-09762-6. arXiv: 1904.12956.

[ARS17] Matthew Amy, Martin Roetteler, and Krysta M. Svore. "Verified compilation of space-efficient reversible circuits". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 3–21. ISBN: 978-3-319-63389-3. DOI: 10.1007/978-3-319-63390-9_1. arXiv: 1603.01635.

[Ato22] Atos. *MyQLM Documentation: The AQASM Format*. 2022. URL: https://myqlm.github.io/aqasm.html (visited on Sept. 1, 2022).

[BACS07] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. "Efficient quantum algorithms for simulating sparse Hamiltonians". In: *Communications in Mathematical Physics* 270 (2 2007), pp. 359–371. DOI: 10.1007/s00220-006-0150-x.

[Bae08] David Baelde. "A Linear Approach to the Proof-Theory of Least and Greatest Fixed Points". PhD thesis. École Polytechnique, Palaiseau, France, 2008.

[Bae12] David Baelde. "Least and greatest fixed points in linear logic". In: *ACM Transactions on Computational Logic* 13.1 (2012), 2:1–2:44. DOI: 10.1145/2071368.2071370. URL: http://doi.acm.org/10.1145/2071368.2071370.

[Bak71] J. W. Bakker. *Recursive Procedures*. Vol. 24. Mathematical Centre Tracts. Mathematisch Centrum Amsterdam, 1971.

[Bak92] Henry G. Baker. "NREVERSAL of fortune - the thermodynamics of garbage collection". In: *International Workshop on Memory Management, IWMM 92* (St. Malo, France, Sept. 17–19, 1992). Ed. by Yves Bekkers and Jacques Cohen. Vol. 637. Lecture Notes in Computer Science. Springer, 1992, pp. 507–524. ISBN: 3-540-55940-X. DOI: 10.1007/BFb0017210.

[Bar10] Stefano Baratella. "Quantum coherent spaces and linear logic". In: *RAIRO Theoretical Informatics Applications* 44.4 (2010), pp. 419–441. DOI: 10.1051/ita/2010021.

[Bar84] Henk P. Barendregt. *The Lambda-Calculus, its Syntax and Semantics*. 2nd ed. Vol. 103. Studies in Logic and the Foundation of Mathematics. North Holland, 1984. ISBN: 0-444-86748-1.

[BB17] Jacob Biamonte and Ville Bergholm. "Tensor Networks in a Nutshell". To appear in Contemporary Physics. Draft available as arXiv:1708.00006. 2017. arXiv: 1708.00006v1.

[BB83] Edwin F. Beckenbach and Richard Bellman. *Inequalities*. Fourth. Vol. 30. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer-Verlag, 1983.

[BBCD+95] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. "Elementary gates for quantum computation". In: *Physical Review A* 52 (5 Nov. 1995), pp. 3457–3467. DOI: 10.1103/PhysRevA.52.3457.

[BBGV20] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin T. Vechev. "Silq: a high-level quantum language with safe uncomputation and intuitive semantics". In: *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI'20* (London, UK). Ed. by Alastair F. Donaldson and Emina Torlak. ACM, 2020, pp. 286–300. DOI: 10.1145/3385412.3386007.

[BBHP92] Nick Benton, Gavin M. Bierman, Martin Hyland, and Valeria C. V. de Paiva. "Linear lambda-calculus and categorical models revisited". In: *Computer Science Logic, Sixth International Workshop, CSL'92, Selected Papers* (San Miniato, Italy). Ed. by E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. M. Richter. Vol. 702. Lecture Notes in Computer Science. European Association for Computer Science Logic. Springer Verlag, Sept. 1992. ISBN: 978-3-540-56992-3. DOI: 10.1007/3-540-56992-8_6.

[BBPH93] Nick Benton, Gavin M. Bierman, Valeria C. V. de Paiva, and Martin Hyland. "A term calculus for intuitionistic linear logic". In: *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93* (Ultrech, Netherlands). Ed. by Marc Bezem and Jan Friso Groote. Vol. 664. Lecture Notes in Computer Science. Springer Verlag, Mar. 1993, pp. 75–90. ISBN: 3-540-56517-5. DOI: 10.1007/BFb0037099.

[BBVA19] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, and Cyril Allouche. "Synthesizing quantum circuits via numerical optimization". In: *Proceedings of the 19th International Conference on Computational Science, ICCS 2019, Part II* (Faro, Portugal, June 12–14, 2019). Ed. by João M. F. Rodrigues, Pedro J. S. Cardoso, Jânio M. Monteiro, Roberto Lam, Valeria V. Krzhizhanovskaya, Michael Harold Lees, Jack J. Dongarra, and Peter M. A. Sloot. Vol. 11537. Lecture Notes in Computer Science. Springer, 2019, pp. 3–16. ISBN: 978-3-030-22740-1. DOI: 10.1007/978-3-030-22741-8_1. HAL: hal-02174967. arXiv: 2004.07714.

[BBVA20] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, and Cyril Allouche. "Quantum circuits synthesis using Householder transformations". In: *Computer Physics Communications* 248 (2020), p. 107001. DOI: 10.1016/j.cpc.2019.107001. HAL: hal-02545123. arXiv: 2004.07710.

[BBVMA20] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Quantum CNOT circuits synthesis for NISQ architectures using the syndrome decoding problem". In: [RC20], pp. 189–205. DOI: 10.1007/978-3-030-52482-1_11.

[BBVMA21] Timothee Goubault De Brugiere, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Reducing the depth of linear reversible quantum circuits". In: *IEEE Transactions on Quantum Engineering* 2 (2021), p. 3102422. DOI: 10.1109/TQE.2021.3091648. HAL: hal-03553916.

[BBVMA22] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Decoding techniques applied to the compilation of CNOT circuits for NISQ architectures". In: *Science of Computer Programming* 214 (2022), p. 102726. DOI: 10.1016/J.SCICO.2021.102726. HAL: hal-03547113.

[BC13] Thomas Braibant and Adam Chlipala. "Formal verification of hardware synthesis". In: *Proceedings of the 25th International Conference on Computer Aided Verification, CAV 2013* (Saint Petersburg, Russia, July 13–19, 2013). Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 213–228. ISBN: 978-3-642-39798-1. DOI: 10.1007/978-3-642-39799-8_14. arXiv: 1301.4779.

[BCCKS14] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. "Exponential improvement in precision for simulating sparse Hamiltonians". In: *Proceedings of the Symposium on Theory of Computing, STOC 2014* (New York, NY, USA, May 31–June 3, 2014). Ed. by David B. Shmoys. ACM, 2014, pp. 283–292. ISBN: 978-1-4503-2710-7. DOI: 10.1145/2591796.2591854. arXiv: 1312.1414.

[BCK15] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. "Hamiltonian simulation with nearly optimal dependence on all parameters". In: *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015* (Berkeley, CA, USA, Oct. 17, 2014–Oct. 20, 2015). Ed. by Venkatesan Guruswami. IEEE Computer Society, 2015, pp. 792–809. DOI: 10.1109/FOCS.2015.54. arXiv: 1501.01715.

[BCKH+22] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. "Noisy intermediate-scale quantum algorithms". In: *Reviews of Modern Physics* 94 (1 Feb. 2022), p. 015004. DOI: 10.1103/RevModPhys.94.015004.

[BCMS89] Roland Backhouse, Paul Chisholm, Grant Malcolm, and Erik Saaman. "Do-it-yourself type theory". In: *Formal Aspects of Computing* 1.1 (1989), pp. 19–84. DOI: 10.1007/BF01887198.

[BCS03] Stefano Bettelli, Tommaso Calarco, and Luciano Serafini. "Toward an architecture for quantum programming". In: *The European Physical Journal D - Atomic, Molecular and Optical Physics* 25.2 (Aug. 2003), pp. 181–200. DOI: 10.1140/epjd/e2003-00242-2. arXiv: cs/0103009.

[BDHP19] Niel de Beaudrap, Ross Duncan, Dominic Horsman, and Simon Perdrix. "Pauli fusion: a computational model to realise quantum transformations from ZX terms". In: [QPL20a], pp. 85–105. DOI: 10.4204/EPTCS.318.6.

[BDS13] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspective in Logic. Cambridge University Press and ASL, 2013.

[BDS16] David Baelde, Amina Doumane, and Alexis Saurin. "Infinitary proof theory: the multiplicative additive case". In: *Proceedings of the 25th EACSL Annual Conference on Computer Science Logic (CSL'16)* (Marseille, France). Ed. by Jean-Marc Talbot and Laurent Regnier. Vol. 62. LIPIcs. Schloss Dagstuhl Leibniz-Zentrum fuer Informatik, 2016, 42:1–42:17. ISBN: 978-3-95977-022-4. DOI: 10.4230/LIPIcs.CSL.2016.42. URL: http://www.dagstuhl.de/dagpub/978-3-95977-022-4.

[Bea03] Stephane Beauregard. "Circuit for Shor's algorithm using $2n+3$ qubits". In: *Quantum Information and Computation* 3.2 (2003), pp. 175–185. DOI: 10.26421/QIC3.2-8.

[Ben00] Charles H. Bennett. "Notes on the history of reversible computation". In: *IBM Journal of Research and Development* 44.1 (2000), pp. 270–278. DOI: 10.1147/rd.441.0270.

[Ben73] Charles H Bennett. "Logical reversibility of computation". In: *IBM Journal of Research and Development* 17.6 (1973), pp. 525–532. DOI: 10.1147/rd.176.0525.

[Ben89] Charles H. Bennett. "Time/space trade-offs for reversible computation". In: *SIAM Journal on Computing* 18.4 (1989), pp. 766–776. DOI: 10.1137/0218053.

[Ben94a] Nick Benton. *A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models*. Tech. rep. UCAM-CL-TR-352. 65 pages. Computer Science department, Cambridge University, 1994.

[Ben94b] Nick Benton. "A mixed linear and non-linear logic: proofs, terms and models (extended abstract)". In: *Computer Science Logic, Eighth International Workshop, CSL'94, Selected Papers* (Kazimierz, Poland). Ed. by Leszek Pacholski and Jerzy Tiuryn. Vol. 933. Lecture Notes in Computer Science. European Association for Computer Science Logic. Springer Verlag, Sept. 1994, pp. 121–135. ISBN: 3-540-60017-5. DOI: 10.1007/BFb0022251.

[Ber00] Gérard Berry. "The foundations of Esterel". In: *Proof, Language, and Interaction, Essays in Honour of Robin Milner*. Ed. by Gordon D. Plotkin, Colin Stirling, and Mads Tofte. The MIT Press, 2000, pp. 425–454. ISBN: 978-0-262-16188-6. DOI: 10.7551/mitpress/5641.003.0021.

[BFMP11] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. "Why3: shepherd your herd of provers". In: *Proceedings of Boogie 2011: First International Workshop on Intermediate Verification Languages* (Wroclaw, Poland). 2011, pp. 53–64. HAL: hal-00790310.

[BFPS81] Wilfried Buchholz, Solomon Feferman, Wolfram Pohlers, and Wilfried Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*. Vol. 897. Lecture Notes in Mathematics. Springer-Verlag, 1981.

[BG89] Val Breazu-Tannen and Jean H. Gallier. "Polymorphic rewriting conserves algebraic strong normalization and confluence". In: *Proceedings of the 16th International Colloquium on Automata, Languages and Programming, ICALP'89* (Stresa, Italy, July 11–15, 1989). Ed. by Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca. Vol. 372. Lecture Notes in Computer Science. Springer, 1989, pp. 137–150. ISBN: 3-540-51371-X. DOI: 10.1007/BFb0035757.

[BG91] Val Breazu-Tannen and Jean H. Gallier. "Polymorphic rewriting conserves algebraic strong normalization". In: *Theoretical Computer Science* 83.1 (1991), pp. 3–28. DOI: 10.1016/0304-3975(91)90037-3.

[BG94] Val Breazu-Tannen and Jean H. Gallier. "Polymorphic rewriting conserves algebraic confluence". In: *Information and Computation* 114.1 (1994). Available as U. Penn. Tech. Report MS-CIS-90-37., pp. 1–29. DOI: 10.1006/inco.1994.1078.

[BH20] Niel de Beaudrap and Dominic Horsman. "The ZX calculus is a language for surface code lattice surgery". In: *Quantum* 4 (2020), p. 218. arXiv: 1704.08670.

[BHYYZ20] Gilles Barthe, Justin Hsu, Mingsheng Ying, Nengkun Yu, and Li Zhou. "Relational proofs for quantum programs". In: *Proceedings of the ACM on Programming Languages* 4.POPL (2020), 21:1–21:29. DOI: 10.1145/3371089.

[Bie93] Gavin M. Bierman. "On Intuitionistic Linear Logic". Available as Technical Report 346, August 1994. PhD thesis. England, UK.: Computer Science department, Cambridge University, 1993.

[BISG+20] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M. Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, Keri McKiernan, Johannes Jakob Meyer, Zeyue Niu, Antal Száva, and Nathan Killoran. "PennyLane: Automatic differentiation of hybrid quantum-classical computations". 2020. arXiv: 1811.04968.

[BK18] Miriam Backens and Aleks Kissinger. "ZH: a complete graphical calculus for quantum computations involving classical non-linearity". In: [QPL19], pp. 23–42. DOI: 10.4204/EPTCS.287.2.

[BKN15] Jaap Boender, Florian Kammüller, and Rajagopal Nagarajan. "Formalization of quantum protocols using coq". In: *Proceedings of the 12th International Workshop on Quantum Physics and Logic, QPL 2015* (Oxford, UK). Ed. by Chris Heunen, Peter Selinger, and Jamie Vicary. Vol. 195. Electronic Proceedings in Theoretical Computer Science. 2015, pp. 71–83. DOI: 10.4204/EPTCS.195.6.

[Blu96] Richard F. Blute. "Hopf algebras and linear logic". In: *Mathematical Structures in Computer Science* 6.2 (1996), pp. 189–212. DOI: 10.1017/S0960129500000943.

[BM04] Stephen S. Bullock and Igor L. Markov. "Asymptotically optimal circuits for arbitrary n-qubit diagonal computations". In: *Quantum Information and Computation* 4.1 (2004), pp. 27–47.

[BM07] David Baelde and Dale Miller. "Least and greatest fixed points in linear logic". In: *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, LPAR'07* (Yerevan, Armenia). Ed. by Nachum Dershowitz and Andrei Voronkov. Vol. 4790. Lecture Notes in Computer Science. Springer, 2007, pp. 92–106. ISBN: 978-3-540-75558-6. DOI: 10.1007/978-3-540-75560-9_9.

[BM87] Val Breazu-Tannen and Albert R. Meyer. "Computable values can be classical". In: *Proceedings of the 14th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'87* (Munich, West Germany). ACM, 1987, pp. 238–245. ISBN: 0-89791-215-2. DOI: 10.1145/41625.41646.

[BN36] Garrett Birkhoff and John Von Neumann. "The logic of quantum mechanics". In: *Annals of Mathematics* 37.4 (1936), pp. 823–843. DOI: 10.2307/1968621.

[BPS93a] Richard F. Blute, Prakash Panangaden, and Robert A.G. Seely. "Fock Space: A Model of Linear Exponential Types". Manuscript, revised version of [BPS93b]. 1993.

[BPS93b] Richard F. Blute, Prakash Panangaden, and Robert A.G. Seely. "Holomorphic models of exponential types in linear logic". In: [MFPS93], pp. 474–512.

[BPV21] Agustín Borgna, Simon Perdrix, and Benoît Valiron. "Hybrid quantum-classical circuit simplification with the ZX-calculus". In: *Proceedings of the 19th Asian Symposium on Programming Languages and Systems, APLAS 2021* (Chicago, IL, USA (Online Conference), Oct. 17–18, 2021). Ed. by Hakjoo Oh. Vol. 13008. Lecture Notes in Computer Science. Springer, 2021, pp. 121–139. DOI: 10.1007/978-3-030-89051-3_8. arXiv: 2109.06071.

[BR72] J. W. de Bakker and Willem P. de Roever. "A calculus for recursive program schemes". In: *Automata, Languages and Programming: Proceedings of a Symposium Organized by IRIA* (Rocquencourt, France, 1972). Ed. by Maurice Nivat. Also found as a Technical Report of *Stichting Mathematisch Centrum*, CWI. North-Holland, Amsterdam, 1972, pp. 167–196. ISBN: 0-7204-2074-1. URL: https://ir.cwi.nl/pub/9145/ (visited on Aug. 11, 2022).

[Bre88] Val Breazu-Tannen. "Combining algebra and higher-order types". In: *Proceedings of the Third Annual Symposium on Logic in Computer Science, LICS '88* (Edinburgh, Scotland, UK, July 5–8, 1988). IEEE Computer Society, 1988, pp. 82–90. ISBN: 0-8186-0853-6. DOI: 10.1109/LICS.1988.5103.

[Bro70] C. G. Broyden. "The convergence of a class of double-rank minimization algorithms". In: *IMA Journal of Applied Mathematics* 6.1 (1970), pp. 76–90.

[Bru20] Timothée Goubault de Brugière. "Methods for optimizing the synthesis of quantum circuits". Thèse de Doctorat. Université Paris-Saclay, 2020. HAL: tel-03127089.

[BSDCM19] Debjyoti Bhattacharjee, Mathias Soeken, Srijit Dutta, Anupam Chattopadhyay, and Giovanni De Micheli. "Reversible pebble games for reducing qubits in hierarchical quantum circuit synthesis". In: *Proceedings of the 49th International Symposium on Multiple-Valued Logic, ISMVL 2019* (Fredericton, NB, Canada, May 21–23, 2019). IEEE, 2019, pp. 102–107. ISBN: 978-1-7281-0092-0. DOI: 10.1109/ISMVL.2019.00026.

[BT04] Patrick Baillot and Kazushige Terui. "Light types for polynomial time computation in lambda-calculus". In: [LICS04], pp. 266–275. DOI: 10.1109/LICS.2004.1319621. HAL: hal-00003468. arXiv: cs/0402059.

[Bur69] Rod M. Burstall. "Proving properties of programs by structural induction". In: *The Computer Journal* 12.1 (1969), pp. 41–48. DOI: 10.1093/comjnl/12.1.41.

[Bur74] Rod M. Burstall. "Program proving as hand simulation with a little induction". In: *Proceedings of the 6th IFIP Congress on Information Processing* (Stockholm, Sweden, Aug. 5–10, 1974). Ed. by Jack L. Rosenfeld. North-Holland, 1974, pp. 308–312. ISBN: 0-7204-2803-3.

[BW15] Xiaoning Bian and Quanlong Wang. "Graphical calculus for qutrit systems". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 98-A.1 (2015), pp. 391–399. DOI: 10.1587/transfun.E98.A.391.

[CABB+21] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. "Variational quantum algorithms". In: *Nature Reviews Physics* 3.9 (2021), pp. 625–644. DOI: 10.1038/s42254-021-00348-9.

[CBBPV21] Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. "An automated deductive verification framework for circuit-building quantum programs". In: *Proceedings of the 30th European Symposium on Programming Languages and Systems, ESOP 2021* (Luxembourg City, Luxembourg, Mar. 27–Apr. 1, 2021). Ed. by Nobuko Yoshida. Vol. 12648. Lecture Notes in Computer Science. Springer, 2021, pp. 148–177. ISBN: 978-3-030-72018-6. DOI: 10.1007/978-3-030-72019-3_6. arXiv: 2003.05841.

[CBLVVX21] Christophe Chareton, Sébastien Bardin, Dongho Lee, Benoît Valiron, Renaud Vilmart, and Zhaowei Xu. "Formal Methods for Quantum Programs: A Survey". Draft, to appear as a book chapter. 2021. arXiv: 2109.06493.

[CBSNG19] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. "Validating quantum computers using randomized model circuits". In: *Physical Review A* 100 (3 Sept. 2019), p. 032328. DOI: 10.1103/PhysRevA.100.032328. arXiv: 1811.12926.

[CCDFGS03] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. "Exponential algorithmic speedup by a quantum walk". In: *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, STOC'03* (San Diego, CA, USA, June 2003). Ed. by Lawrence L. Larmore and Michel X. Goemans. ACM, 2003, pp. 59–68. ISBN: 1-58113-674-9. DOI: 10.1145/780542.780552.

[CD07] Bob Coecke and Ross Duncan. "A graphical calculus for quantum observables". Historical note from Bob Coecke: First paper containing ZX-diagrams. Rejected from QIP with reports like: "nice pictures, so what?". 2007. URL: http://www.cs.ox.ac.uk/people/bob.coecke/GreenRed.pdf (visited on Aug. 24, 2022).

[CD08] Bob Coecke and Ross Duncan. "Interacting quantum observables". In: *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008), Part II* (Reykjavik, Iceland, July 7–11, 2008). Ed. by Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz. Vol. 5126. Lecture Notes in Computer Science. Springer, 2008, pp. 298–310. DOI: 10.1007/978-3-540-70583-3_25.

[CD11] Bob Coecke and Ross Duncan. "Interacting quantum observables: categorical algebra and diagrammatics". In: *New Journal of Physics* 13.4 (Apr. 2011), p. 043016. DOI: 10.1088/1367-2630/13/4/043016. arXiv: 0906.4725.

[CD22] Andrea Colledan and Ugo Dal Lago. "On Dynamic Lifting and Effect Typing in Circuit Description Languages (Extended Version)". Presented at TYPES 2022. 2022. arXiv: 2202.07636.

[CD93] Charles Consel and Olivier Danvy. "Tutorial notes on partial evaluation". In: *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'93* (Charleston, South Carolina, USA). Ed. by Mary S. Van Deusen and Bernard Lang. ACM Press, 1993, pp. 493–501. ISBN: 0-89791-560-7. DOI: 10.1145/158511.158707.

[CDP08a] G. Chiribella, G. M. D'Ariano, and P. Perinotti. "Quantum circuit architecture". In: *Physical Review Letters* 101 (6 Aug. 2008), p. 060401. DOI: 10.1103/PhysRevLett.101.060401. arXiv: 0712.1325.

[CDP08b] G. Chiribella, G. M. D'Ariano, and P. Perinotti. "Transforming quantum operations: quantum supermaps". In: *EPL (Europhysics Letters)* 83.3 (2008), p. 30004. DOI: 10.1209/0295-5075/83/30004. arXiv: 0804.0180.

[CDP09] Giulio Chiribella, Giacomo Mauro D'Ariano, and Paolo Perinotti. "Theoretical framework for quantum networks". In: *Physical Review A* 80 (2 Aug. 2009), p. 022339. DOI: 10.1103/PhysRevA.80.022339.

[CDPV13] G. Chiribella, G. M. D'Ariano, P. Perinotti, and B. Valiron. "Quantum computations without definite causal structure". In: *Physical Review A* 88 (2013), p. 022318. DOI: 10.1103/PhysRevA.88.022318. arXiv: 0912.0195.

[CFC58] Haskell H. Curry, Robert Feys, and William Craig. *Combinatory Logic*. Vol. 22. Studies in Logic and the Foundations of Mathematics. North-Holland, 1958.

[CFM17] Frederic T. Chong, Diana Franklin, and Margaret Martonosi. "Programming languages and compiler design for realistic quantum hardware". In: *Nature* 549 (7671 2017), pp. 180–187. DOI: 10.1038/nature23459.

[Cha23] Kostia Chardonnet. "Vers une correspondance de Curry-Howard pour le calcul quantique". Thèse de Doctorat. Université Paris-Saclay, 2023.

[CHMPV22] Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. "LOv-calculus: a graphical language for linear optical quantum circuits". In: *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*. Ed. by Stefan Szeider, Robert Ganian, and Alexandra Silva. Vol. 241. LIPIcs. 2022, 35:1–35:16. DOI: 10.4230/LIPICS.MFCS.2022.35. URL: https://doi.org/10.4230/LIPIcs.MFCS.2022.35.

[CHMPV23a] Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. "A complete equational theory for quantum circuits". In: *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*. IEEE, 2023, pp. 1–13. DOI: 10.1109/LICS56636.2023.10175801. HAL: hal-03926757.

[CHMPV23b] Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. "A Complete Equational Theory for Quantum Circuits". Presentation accepted at the 18th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2023), in Aveiro, Portugal. 2023. HAL: hal-04318291v1.

[Cho75] Man-Duen Choi. "Completely positive linear maps on complex matrices". In: *Linear Algebra and its Applications* 10.3 (1975), pp. 285–290.

[CHP19] Titouan Carette, Dominic Horsman, and Simon Perdrix. "SZX-calculus: scalable graphical quantum reasoning". In: *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019* (Aachen, Germany, Aug. 26–30, 2019). Ed. by Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen. Vol. 138. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 55:1–55:15. ISBN: 978-3-95977-117-7. DOI: 10.4230/LIPIcs.MFCS.2019.55.

[Chu36] Alonzo Church. "An unsolvable problem of elementary number theory". In: *American Journal of Mathematics* 58.2 (1936), pp. 345–363. DOI: 10.2307/2371045.

[CJ20] Titouan Carette and Emmanuel Jeandel. "A recipe for quantum graphical languages". In: *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020* (Saarbrücken, Germany (Virtual Conference), July 8, 2022–July 11, 2020). Ed. by Artur Czumaj, Anuj Dawar, and Emanuela Merelli. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 118:1–118:17. DOI: 10.4230/LIPIcs.ICALP.2020.118.

[CJPV19] Titouan Carette, Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. "Completeness of graphical languages for mixed states quantum mechanics". In: *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019* (Patras, Greece, July 9–12, 2019). Ed. by Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 108:1–108:15. ISBN: 978-3-95977-109-2. DOI: 10.4230/LIPIcs.ICALP.2019.108. URL: http://www.dagstuhl.de/dagpub/978-3-95977-109-2.

[CJS13] B. D. Clader, B. C. Jacobs, and C. R. Sprouse. "Preconditioned quantum linear system algorithm". In: *Physical Review Letters* 110 (25 2013), p. 250504. DOI: 10.1103/PhysRevLett.110.250504. ARXIV: 1301.2340.

[CJV93] A. Chatterjee, J. M. Jin, and J. L. Volakis. "Edge-based finite elements and vector abc's applied to 3D scattering". In: *IEEE Transactions on Antennas and Propagation* 41.2 (1993).

[CK17] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017. ISBN: 9781316219317. DOI: 10.1017/9781316219317.

[CK97] Roberto Di Cosmo and Delia Kesner. "Strong normalization of explicit substitutions via cut elimination in proof nets (extended abstract)". In: *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science, LICS'97* (Warsaw, Poland, June 29–July 2, 1997). IEEE Computer Society, 1997, pp. 35–46. ISBN: 0-8186-7925-5. DOI: 10.1109/LICS.1997.614927.

[CKRZH18] Nicholas Chancellor, Aleks Kissinger, Joschka Roffe, Stefan Zohren, and Dominic Horsman. "Graphical Structures for Design and Verification of Quantum Error Correction". Draft. 2018.

[CL02] J. Robin B. Cockett and Stephen Lack. "Restriction categories I: categories of partial maps". In: *Theoretical Computer Science* 270.1 (2002), pp. 223–259. DOI: 10.1016/S0304-3975(00)00382-0.

[CL03] J. Robin B. Cockett and Stephen Lack. "Restriction categories ii: partial map classification". In: *Theoretical Computer Science* 294.1 (2003), pp. 61–102. DOI: 10.1016/S0304-3975(01)00245-6.

[CL07] Robin Cockett and Stephen Lack. "Restriction categories III: colimits, partial limits and extensivity". In: *Mathematical Structures in Computer Science* 17.4 (2007), pp. 775–817. DOI: 10.1017/S0960129507006056.

[Cla01] Koen Claessen. "Embedded Languages for Describing and Verifying Hardware". Doktorsavhandlingar. Chalmers University of Technology, Gothenburg, Sweden, 2001. ISBN: 91-7291-014-3.

[CLV21] Kostia Chardonnet, Louis Lemonnier, and Benoît Valiron. "Categorical semantics of reversible pattern-matching". In: [MFPS21], pp. 18–33. DOI: 10.4204/EPTCS.351.2.

[Coe04] Bob Coecke. "Quantum information-flow, concretely, abstractly". In: [QPL04], pp. 57–73.

[Cop17] B. Jack Copeland. "The Church-Turing thesis". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2020 Edition. 2017. URL: https://plato.stanford.edu/archives/sum2020/entries/church-turing/ (visited on Sept. 6, 2021).

[CP90] T. Coquand and C. Paulin. "Inductively defined types". In: [MM90], pp. 50–66.

[CPV13] Bob Coecke, Dusko Pavlovic, and Jamie Vicary. "A new description of orthogonal bases". In: *Mathematical Structures in Computer Science* 23.3 (2013), pp. 555–567. DOI: 10.1017/S0960129512000047.

[CS16] Jacques Carette and Amr Sabry. "Computing with semirings and weak rig groupoids". In: *Proceedings of the 25th European Symposium on Programming Languages and Systems (ESOP'16)* (Eindhoven, The Netherlands). Ed. by Peter Thiemann. Vol. 9632. Lecture Notes in Computer Science. Springer, Apr. 2016, pp. 123–148. DOI: 10.1007/978-3-662-49498-1_6.

[CSV20] Kostia Chardonnet, Alexis Saurin, and Benoît Valiron. "Toward a Curry-Howard equivalence for linear, reversible computation - work-in-progress". In: [RC20], pp. 144–152. DOI: 10.1007/978-3-030-52482-1_8. HAL: hal-03103455.

[CSV21] Kostia Chardonnet, Alexis Saurin, and Benoît Valiron. "Towards a Curry-Howard correspondence for linear, reversible computation". In: *Proceedings of the 5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)* (Rome (virtual), Italy). 2021. HAL: lirmm-03271484.

[CSV23] Kostia Chardonnet, Alexis Saurin, and Benoît Valiron. "A curry-howard correspondence for linear, reversible computation". In: *31st EACSL Annual Conference on Computer Science Logic, CSL 2023* (Warsaw, Poland, Feb. 13–16, 2023). Ed. by Bartek Klin and Elaine Pimentel. Vol. 252. LIPIcs. 2023, 13:1–13:18. DOI: 10.4230/LIPICS.CSL.2023.13.

[CTV23] Théodore Chapuis-Chkaiban, Zeno Toffano, and Benoît Valiron. "On new pagerank computation methods using quantum computing". In: *Quantum Information Processing* 22.3 (2023), p. 138. DOI: 10.1007/S11128-023-03856-Y. HAL: hal-04056045.

[CV20] Pierre Clairambault and Marc de Visme. "Full abstraction for the quantum lambda-calculus". In: *Proceedings of the ACM on Programming Languages* 4.POPL (2020), 63:1–63:28. DOI: 10.1145/3371131.

[CVCC23] Matheus Capela, Harshit Verma, Fabio Costa, and Lucas C. Céleri. "Reassessing thermodynamic advantage from indefinite causal order". In: *Physical Review A* 107 (6 2023), p. 062208. DOI: 10.1103/PhysRevA.107.062208.

[CVV21] Kostia Chardonnet, Benoît Valiron, and Renaud Vilmart. "Geometry of interaction for ZX diagrams". In: *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021* (Tallinn, Estonia). Ed. by Filippo Bonchi and Simon J. Puglisi. Vol. 202. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2021, 30:1–30:16. ISBN: 978-3-95977-201-3. DOI: 10.4230/LIPIcs.MFCS.2021.30.

[CVVV22] Kostia Chardonnet, Marc de Visme, Benoît Valiron, and Renaud Vilmart. "The Many-Worlds Calculus: Representing Quantum Control". 2022.

[CW96] Edmund M. Clarke and Jeannette M. Wing. "Formal methods: state of the art and future directions". In: *ACM Computing Surveys* 28.4 (1996), pp. 626–643. DOI: 10.1145/242223.242257.

[Cyb01] George Cybenko. "Reducing quantum computations to elementary unitary operations". In: *Computing in Science & Engineering* 3.2 (2001), pp. 27–32. DOI: 10.1109/5992.908999.

[Dan90] Vincent Danos. "La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du Lambda-calcul)". Thèse de Doctorat en Mathématiques. Université Paris 7, 1990.

[DD17] Alejandro Díaz-Caro and Gilles Dowek. "Typing quantum superpositions and measurement". In: *Proceedings of the 6th International Conference on the Theory and Practice of Natural Computing (TPNC'17)* (Prague, Czech Republic). Ed. by Carlos Martín-Vide, Roman Neruda, and Miguel A. Vega-Rodríguez. Vol. 10687. Lecture Notes in Computer Science. Springer, 2017, pp. 281–293. DOI: 10.1007/978-3-319-71069-3_22.

[DE11] Vincent Danos and Thomas Ehrhard. "Probabilistic coherence spaces as a model of higher-order probabilistic computation". In: *Information and Computation* 209.6 (2011), pp. 966–991. DOI: 10.1016/j.ic.2011.02.001.

[Del08a] Yannick Delbecque. "A quantum game semantics for the measurement calculus". In: [QPL08], pp. 33–48.

[Del08b] Yannick Delbecque. "Quantum Games as Quantum Types". PhD thesis. McGill University, 2008.

[Del11] Yannick Delbecque. "Game semantics for quantum data". In: [QPL11], pp. 41–57. DOI: 10.1016/j.entcs.2011.01.005.

[Deu85] David Deutsch. "Quantum theory, the Church-Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London A* 400.1818 (1985), pp. 97–117. DOI: 10.1098/rspa.1985.0070.

[DG02] Maria Luisa Dalla Chiara and Roberto Giuntini. "Quantum logics". In: *Handbook of Philosophical Logic*. Ed. by Dov M. Gabbay and F. Guenthner. 2nd ed. Vol. 6. Springer, Dordrecht, 2002, pp. 129–228. ISBN: 978-1-4020-0583-1. DOI: 10.1007/978-94-017-0460-1_2.

[DG08] Nachum Dershowitz and Yuri Gurevich. "A natural axiomatization of computability and proof of Church's thesis". In: *Bulletin of Symbolic Logic* 14.3 (2008), pp. 299–350. DOI: 10.2178/bsl/1231081370.

[DGMV19] Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel, and Benoît Valiron. "Realizability in the unitary sphere". In: [LICS19], pp. 1–13. DOI: 10.1109/LICS.2019.8785834. HAL: hal-02175168. arXiv: 1904.08785.

[DH02] Vincent Danos and Russel S. Harmer. "Probabilistic game semantics". In: *ACM Transactions on Computational Logic* 3.3 (2002), pp. 359–382. DOI: 10.1145/507382.507385.

[DHKK95] Peter Deussen, A. Hansmann, Thomas Käufl, and Stefan Klingenbeck. "The verification system Tatzelwurm". In: *KORSO - Methods, Languages, and Tools for the Construction of Correct Software*. Ed. by Manfred Broy and Stefan Jähnichen. Vol. 1009. Lecture Notes in Computer Science. Springer, 1995, pp. 285–298. ISBN: 3-540-60589-4. DOI: 10.1007/BFb0015468.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976. ISBN: 013215871X. URL: https://www.worldcat.org/oclc/01958445.

[DKPW20] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. "Graph-theoretic simplification of quantum circuits with the ZX-calculus". In: *Quantum* 4 (2020), p. 279. ISSN: 2521-327X. DOI: 10.22331/q-2020-06-04-279.

[DKRS06] Thomas G. Draper, Samuel A. Kutin, Eric M. Rains, and Krysta M. Svore. "A logarithmic-depth quantum carry-lookahead adder". In: *Quantum Information and Computation* 6.4–5 (2006), pp. 351–369. arXiv: quant-ph/0406142.

[DL13] Ross Duncan and Maxime Lucas. "Verifying the Steane code with Quantomatic". In: *Proceedings of the 10th International Workshop on Quantum Physics and Logic, QPL 2013* (Castelldefels (Barcelona), Spain, July 17–19, 2013). Ed. by Bob Coecke and Matty J. Hoban. Vol. 171. EPTCS. 2013, pp. 33–49. DOI: 10.4204/EPTCS.171.4.

[Dou17] Amina Doumane. "On the Infinitary Proof Theory of Logics with Fixed Points". Thèse de doctorat. Université Paris Diderot, 2017.

[Dou92] Daniel J. Dougherty. "Adding algebraic rewriting to the untyped lambda calculus". In: *Information and Computation* 101.2 (1992), pp. 251–267. DOI: 10.1016/0890-5401(92)90064-M.

[Dow12] Gilles Dowek. "Around the physical Church-Turing thesis: cellular automata, formal languages, and the principles of quantum theory". In: *Proceedingsof the 6th International Conference on Language and Automata Theory and Applications, LATA 2012* (A Coruña, Spain, Mar. 5–9, 2012). Ed. by Adrian-Horia Dediu and Carlos Martín-Vide. Vol. 7183. Lecture Notes in Computer Science. Springer, 2012, pp. 21–37. ISBN: 978-3-642-28331-4. DOI: 10.1007/978-3-642-28332-1_3.

[dP04] Ellie d'Hondt and Prakash Panangaden. "Quantum weakest preconditions". In: [QPL04], pp. 75–90.

[dP06] Ellie d'Hondt and Prakash Panangaden. "Quantum weakest preconditions". In: *Mathematical Structures in Computer Science* 16.3 (2006), pp. 429–451. DOI: 10.1017/S0960129506005251. arXiv: quant-ph/0501157.

[DP08] Yannick Delbecque and Prakash Panangaden. "Game semantics for quantum stores". In: *Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics, MFPS XXIV* (Philadelphia, PA, USA). Ed. by A. Bauer and M. Mislove. Vol. 218. Electronic Notes in Theoretical Computer Science. May 2008, pp. 153–170. DOI: 10.1016/j.entcs.2008.10.010.

[DP10] Ross Duncan and Simon Perdrix. "Rewriting measurement-based quantum computations with generalised flow". In: *Proceedings of the 37th International Colloquium on Automata, Languages and Programming, ICALP'10, Part II* (Bordeaux, France). Ed. by Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis. Vol. 6199. Lecture Notes in Computer Science. Springer, 2010, pp. 285–296. DOI: 10.1007/978-3-642-14162-1_24.

[dP95] Ugo de'Liguoro and Adolfo Piperno. "Non deterministic extensions of untyped lambda-calculus". In: *Information and Computation* 122.2 (1995), pp. 149–177. DOI: 10.1006/inco.1995.1145.

[DPTV10] Alejandro Díaz-Caro, Simon Perdrix, Christine Tasson, and Benoît Valiron. "Equivalence of algebraic lambda-calculi (work in progress)". In: *Pre-Proceedings of the 5th International Workshop on Higher-Order Rewriting (HOR'10), Edinburgh, 14 juillet 2010*. This work has been finalized in [ADPTV14]. 2010, pp. 6–11.

[DR89] Vincent Danos and Laurent Regnier. "The structure of multiplicatives". In: *Archive for Mathematical Logic* 28.3 (1989), pp. 181–203. DOI: 10.1007/BF01622878.

[DR99] Vincent Danos and Laurent Regnier. "Reversible, irreversible and optimal lambda-machines". In: *Theoretical Computer Science* 227.1-2 (1999), pp. 79–97. DOI: 10.1016/S0304-3975(99)00049-3.

[Dun09] Ross Duncan. "Generalized proof-nets for compact categories with biproducts". In: [GM09]. Chap. 3, pp. 70–134. DOI: 10.1017/CBO9781139193313.004. arXiv: 0903.5154.

[Dun13] Ross Duncan. "A graphical approach to measurement-based quantum computing". In: *Quantum Physics and Linguistics: A Compositional, Diagrammatic Discourse*. Ed. by Chris Heunen, Mehrnoosh Sadrzadeh, and Edward Grefenstette. Oxford University Press, 2013. Chap. 3. ISBN: 9780199646296. DOI: 10.1093/acprof:oso/9780199646296.003.0003. arXiv: 1203.6242.

[Dur02] Jérome Durand-Lose. "Computing inside the billiard ball model". In: [Ada02], pp. 135–160. URL: http://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose/Recherche/Publications/2002_BBM_book.pdf.

[Dyb91] Peter Dybjer. "Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics". In: ed. by Gérard Huet and Gordon D. Plotkin. Cambridge University Press, 1991, pp. 280–306. ISBN: 9780511569807. DOI: 10.1017/CBO9780511569807.012.

[Dyb94] Peter Dybjer. "Inductive families". In: *Formal Aspects of Computing* 6.4 (1994), pp. 440–465. DOI: 10.1007/BF01211308.

[Ehr02] Thomas Ehrhard. "On köthe sequence spaces and linear logic". In: *Mathematical Structures in Computer Science* 12.5 (2002), pp. 579–623. DOI: 10.1017/S0960129502003729.

[Ehr05] Thomas Ehrhard. "Finiteness spaces". In: *Mathematical Structures in Computer Science* 15.4 (2005), pp. 615–646. DOI: 10.1017/S0960129504004645.

[EPT11] Thomas Ehrhard, Michele Pagani, and Christine Tasson. "The computational meaning of probabilistic coherence spaces". In: [LICS11], pp. 87–96. DOI: 10.1109/LICS.2011.29. HAL: hal-00627490.

[ER03] Thomas Ehrhard and Laurent Regnier. "The differential lambda-calculus". In: *Theoretical Computer Science* 309.1-3 (2003), pp. 1–41. DOI: 10.1016/S0304-3975(03)00392-X.

[ER06] Thomas Ehrhard and Laurent Regnier. "Differential interaction nets". In: *Theoretical Computer Science* 364.2 (2006), pp. 166–195. DOI: 10.1016/j.tcs.2006.08.003. HAL: hal-00150274.

[ETP14] Thomas Ehrhard, Christine Tasson, and Michele Pagani. "Probabilistic coherence spaces are fully abstract for probabilistic PCF". In: [POPL14], pp. 309–320. DOI: 10.1145/2535838.2535865.

[FBCH+20] Michael P. Frank, Robert W. Brocato, Thomas M. Conte, Alexander H. Hsia, Anirudh Jain, Nancy A. Missert, Karpur Shukla, and Brian D. Tierney. "Special session: exploring the ultimate limits of adiabatic circuits". In: *38th IEEE International Conference on Computer Design (ICCD)*. 2020, pp. 21–24. DOI: 10.1109/ICCD50377.2020.00018.

[FC22] Giovanni de Felice and Bob Coecke. "Quantum linear optics via string diagrams". In: [QPL23a], pp. 83–100. DOI: 10.4204/EPTCS.394.6.

[FD19] Andrew Fagan and Ross Duncan. "Optimising Clifford circuits with Quantomatic". In: [QPL19], pp. 85–105. DOI: 10.4204/EPTCS.287.5.

[FDDB14] Nicolai Friis, Vedran Dunjko, Wolfgang Dür, and Hans J. Briegel. "Implementing quantum control for unknown subroutines". In: *Physical Review A* 89 (3 2014), p. 030303. DOI: 10.1103/PhysRevA.89.030303. arXiv: 1401.8128.

[Fef70] Solomon Feferman. "Formal theories for transfinite iterations of generalized inductive definitions and some subsystems of analysis". In: *Intuitionism and Proof Theory: Proceedings of the Summer Conference at Buffalo N.Y. 1968*. Ed. by A. Kino, J. Myhill, and R. E. Vesley. Vol. 60. Studies in Logic and the Foundations of Mathematics. North-Holland, 1970, pp. 303–326. DOI: 10.1016/S0049-237X(08)70761-4.

[Fey82] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.7-8 (1982), pp. 467–488.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. Tech. rep. MIT-CTP/4610. MIT, 2014.

[FGGLLP01] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. "A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem". In: *Science* 292.5516 (2001), pp. 472–475. DOI: 10.1126/science.1057726. arXiv: quant-ph/0104129.

[FGGS00] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. *Quantum Computation by Adiabatic Evolution*. Tech. rep. MIT-CTP-2936. MIT, 2000.

[FH65] Richard P. Feynman and A. R. Hibbs. *Quantum Mechanics and Path Integrals*. McGraw-Hill Publishing Company, 1965. ISBN: 0-07-020650-3.

[FHTZ15] Yuan Feng, Ernst Moritz Hahn, Andrea Turrini, and Lijun Zhang. "QPMC: a model checker for quantum programs and protocols". In: *Proceedings of the 20th International Symposium on Formal Methods (FM 2015)* (Oslo, Norway). Ed. by Nikolaj Bjørner and Frank S. de Boer. Vol. 9109. Lecture Notes in Computer Science. Springer, 2015, pp. 265–272. ISBN: 978-3-319-19248-2. DOI: 10.1007/978-3-319-19249-9_17.

[FKRS20] Peng Fu, Kohei Kishida, Neil J. Ross, and Peter Selinger. "A tutorial introduction to quantum circuit programming in dependently typed proto-quipper". In: [RC20], pp. 153–168. DOI: 10.1007/978-3-030-52482-1_9. arXiv: 2005.08396.

[FKRS22a] Peng Fu, Kohei Kishida, Neil J. Ross, and Peter Selinger. "A biset-enriched categorical model for Proto-Quipper with dynamic lifting". In: [QPL23a]. DOI: 10.4204/EPTCS.394.16. arXiv: 2204.13039.

[FKRS22b] Peng Fu, Kohei Kishida, Neil J. Ross, and Peter Selinger. "Proto-Quipper with dynamic lifting". See also the companion paper [FKRS22a]. 2022. arXiv: 2204.13041.

[FKS20] Peng Fu, Kohei Kishida, and Peter Selinger. "Linear dependent type theory for quantum programming languages: extended abstract". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2020* (Saarbrücken, Germany). Ed. by Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller. ACM, July 2020, pp. 440–453. ISBN: 978-1-4503-7104-9. DOI: 10.1145/3373718.3394765. arXiv: 2004.13472.

[Fle70] R. Fletcher. "A new approach to variable metric algorithms". In: *The Computer Journal* 13.3 (1970), pp. 317–322. DOI: 10.1093/comjnl/13.3.317.

[Flo67] Robert W. Floyd. "Assigning meanings to programs". In: *Mathematical Aspects of Computer Science*. Ed. by J. T. Schwartz. Vol. 19. Proceedings of Symposia in Applied Mathematics. AMS, 1967, pp. 19–32. DOI: 10.1090/psapm/019.

[FLY22] Yuan Feng, Sanjiang Li, and Mingsheng Ying. "Verification of distributed quantum programs". In: *ACM Transactions in Computational Logic* 23.3 (2022), 19:1–19:40. DOI: 10.1145/3517145.

[FM07] Jean-Christophe Filliâtre and Claude Marché. "The Why/Krakatoa/Caduceus platform for deductive program verification". In: *Proceedings of the 19th International Conference on Computer Aided Verification, CAV 2007* (Berlin, Germany, July 3–7, 2007). Ed. by Werner Damm and Holger Hermanns. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007, pp. 173–177. ISBN: 978-3-540-73367-6. DOI: 10.1007/978-3-540-73368-3_21. HAL: inria-00270820.

[FP13] Jean-Christophe Filliâtre and Andrei Paskevich. "Why3 – where programs meet provers". In: *Proceedings of the 22nd European Symposium on Programming Languages and Systems, ESOP 2013* (Rome, Italy, Mar. 16–24, 2013). Ed. by Matthias Felleisen and Philippa Gardner. Vol. 7792. Lecture Notes in Computer Science. Springer, 2013, pp. 125–128. ISBN: 978-3-642-37035-9. DOI: 10.1007/978-3-642-37036-6_8. HAL: hal-00789533.

[Fra99] Michael Patrick Frank. "Reversibility for Efficient Computing". PhD thesis. MIT, 1999.

[FT82] Edward Fredkin and Tommaso Toffoli. "Conservative logic". In: *International Journal of Theoretical Physics* 21 (3 1982), pp. 219–253. DOI: 10.1007/BF01857727.

[FTR07] K. Fazel, M. A. Thornton, and J. E. Rice. "ESOP-based Toffoli gate cascade generation". In: *Proceedings of the 2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM 2007* (Victoria, BC, Canada, Aug. 22–24, 2007). IEEE, 2007. ISBN: 978-1-4244-1189-4. DOI: 10.1109/PACRIM.2007.4313212.

[GAJ06] Pallav Gupta, Abhinav Agrawal, and Niraj K. Jha. "An algorithm for synthesis of reversible logic circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.11 (2006), pp. 2317–2330. DOI: 10.1109/TCAD.2006.871622.

[Gan80] Robin Gandy. "Church's thesis and principles for mechanisms". In: *The Kleene Symposium* (Madison, Wisconsin, USA, June 18–24, 1978). Ed. by Jon Barwise, H. Jerome Keisler, and Kenneth Kunen. Vol. 101. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, 1980, pp. 123–148. ISBN: 978-0-444-85345-5. DOI: 10.1016/S0049-237X(08)71257-6.

[Gay06] Simon J. Gay. "Quantum programming languages: survey and bibliography". In: *Mathematical Structures in Computer Science* 16.4 (2006), pp. 581–600. DOI: 10.1017/S0960129506005378.

[GBVMA21] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. "Gaussian elimination versus greedy methods for the synthesis of linear reversible circuits". In: *ACM Transactions on Quantum Computing* 2.3 (2021), p. 11. DOI: 10.1145/3474226. HAL: hal-03547117.

[GD17] Liam Garvie and Ross Duncan. "Verifying the smallest interesting colour code with Quantomatic". In: [QPL18], pp. 147–163. DOI: 10.4204/EPTCS.266.10.

[GE21] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". In: *Quantum* 5 (2021), p. 433. DOI: 10.22331/q-2021-04-15-433. arXiv: 1905.09749.

[GHHNP13] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. "Linear dependent types for differential privacy". In: *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13* (Rome, Italy). Ed. by Roberto Giacobazzi and Radhia Cousot. ACM, 2013, pp. 357–370. ISBN: 978-1-4503-1832-7. DOI: 10.1145/2429069.2429113.

[Ghi07] Dan R. Ghica. "Geometry of synthesis: a structured approach to VLSI design". In: *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007* (Nice, France). Ed. by Martin Hofmann and Matthias Felleisen. ACM, Jan. 2007, pp. 363–375. ISBN: 1-59593-575-4. DOI: 10.1145/1190216.1190269. URL: https://doi.org/10.1145/1190216.1190269.

[Ghi12] Dan R. Ghica. "The geometry of synthesis – how to make hardware out of software". In: *Proceedings of the 11th International Conference on Mathematics of Program Construction, MPC 2012* (Madrid, Spain). Ed. by Jeremy Gibbons and Pablo Nogueira. Vol. 7342. Lecture Notes in Computer Science. Abstract of Invited Talk. Springer, June 2012, pp. 23–24. ISBN: 978-3-642-31112-3. DOI: 10.1007/978-3-642-31113-0_3.

[GHKLMS03] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous Lattices and Domains*. Vol. 93. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2003. ISBN: 0-521-80338-1.

[Gir03] Jean-Yves Girard. "Geometry of interaction IV: the feedback equation". In: *Logic Colloquium '03. Proceeedings of the Annual European Summer Meeting of the Association for Symbolic Logic, held in Helsinki, Finland, August 14–20, 2003.* Ed. by Viggo Stoltenberg-Hansen and Jouko Väänänen. Vol. 24. Lecture Notes in Logic. ASL, 2003, pp. 76–217.

[Gir04] Jean-Yves Girard. "Between logic and quantic – a tract". In: ed. by Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott. Vol. 316. London Mathematical Society Lecture Notes Series. Cambridge University Press, 2004. Chap. 10, pp. 346–381. ISBN: 0-521-60857-0. DOI: 10.1017/CBO9780511550850.011.

[Gir11] Jean-Yves Girard. "Geometry of interaction V: logic in the hyperfinite factor". In: *Theoretical Computer Science* 412.20 (2011), pp. 1860–1883. DOI: 10.1016/j.tcs.2010.12.016.

[Gir86] Jean-Yves Girard. "The system F of variable types, fifteen years later". In: *Theor. Comput. Sci.* 45.2 (1986), pp. 159–192. DOI: 10.1016/0304-3975(86)90044-7.

[Gir87] Jean-Yves Girard. "Linear logic". In: *Theoretical Computer Science* 50.1 (1987), pp. 1–101. DOI: 10.1016/0304-3975(87)90045-4.

[Gir88] Jean-Yves Girard. "Normal functors, power series and lambda-calculus". In: *Annals of Pure and Applied Logic* 37.2 (1988), pp. 129–177. DOI: 10.1016/0168-0072(88)90025-5.

[Gir89] Jean-Yves Girard. "Geometry of interaction I: interpretation of system F". In: *Logic Colloquium '88. Proceedings of the Colloquium Held in Padova, Italy, August 22–31, 1988.* Ed. by R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo. Vol. 127. Studies in Logic and the Foundations of Mathematics. North-Holland, 1989, pp. 221–260. DOI: 10.1016/S0049-237X(08)70271-4.

[Gir90] Jean-Yves Girard. "Geometry of interaction II: deadlock-free algorithms". In: [MM90], pp. 76–93. DOI: 10.1007/3-540-52335-9_49.

[Gir95a] Jean-Yves Girard. "Geometry of interaction III: accommodating the additives". In: [GLR95], pp. 329–389.

[Gir95b] Jean-Yves Girard, ed. *La Machine de Turing*. Vol. 131. Points Sciences. Contains [Tur36] and [Tur50] in integrality, with comments. Editions du Seuil, 1995. ISBN: 2-02-036928-1.

[Gir98] Jean-Yves Girard. "Light linear logic". In: *Information and Computation* 143.2 (1998), pp. 175–204. DOI: 10.1006/inco.1998.2700.

[Gir99] Jean-Yves Girard. "Coherent Banach spaces: a continuous denotational semantics". In: *Theoretical Computer Science* 227.1-2 (1999), pp. 275–297. DOI: 10.1016/S0304-3975(99)00056-0.

[GKY19] Robert Glück, Robin Kaarsgaard, and Tetsuo Yokoyama. "Reversible programs have reversible semantics". In: *Formal Methods. FM 2019 International Workshops - Porto, Portugal, October 7-11, 2019, Revised Selected Papers, Part II.* Ed. by Emil Sekerinski, Nelma Moreira, José N. Oliveira, Daniel Ratiu, Riccardo Guidotti, Marie Farrell, Matt Luckcuck, Diego Marmsoler, José Campos, Troy Astarte, Laure Gonnord, Antonio Cerone, Luis Couto, Brijesh Dongol, Martin Kutrib, Pedro Monteiro, and David Delmas. Vol. 12233. Lecture Notes in Computer Science. Springer, 2019, pp. 413–427. ISBN: 978-3-030-54996-1. DOI: 10.1007/978-3-030-54997-8_26.

[GLR95] Jean-Yves Girard, Yves Lafont, and Laurent Regnier, eds. *Advances in Linear Logic*. Vol. 222. London Mathematical Society Lecture Note Series. Cambridge University Press, 1995. ISBN: 0-521-55961-8.

[GLRSV12] Alexander S. Green, Peter L. Lumsdaine, Neil J. Ross, Peter Selinger, and Benoî Valiron. *Report on the Quipper language, version 0.3, with GFI algorithm implementations (Updated for revision 0.3-4)*. Report to IARPA, for official use only. 2012.

[GLRSV13a] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. "An introduction to quantum programming in quipper". In: [RC13], pp. 110–124. DOI: 10.1007/978-3-642-38986-3_10. arXiv: 1304.5485.

[GLRSV13b] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. "Quipper: a scalable quantum programming language". In: *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'13* (Seattle, WA, USA). Ed. by Hans-Juergen Boehm and Cormac Flanagan. ACM, 2013, pp. 333–342. ISBN: 978-1-4503-2014-6. DOI: 10.1145/2491956.2462177. arXiv: 1304.3390.

[GLT90] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. 2nd ed. Vol. 7. Cambridge Tracts In Theoretical Computer Science. Cambridge University Press, 1990. ISBN: 0-521-37181-3. URL: http://www.cs.man.ac.uk/~pt/stable/Proofs+Types.html.

[GM01] Stefano Guerrini and Andrea Masini. "Parsing MELL proof nets". In: *Theoretical Computer Science* 254.1-2 (2001), pp. 317–335. DOI: 10.1016/S0304-3975(99)00299-6.

[GM09] Simon Gay and Ian Mackie, eds. *Semantic Techniques in Quantum Computation*. Cambridge University Press, 2009. ISBN: 978-0-521-51374-6.

[GNP08] Simon J. Gay, Rajagopal Nagarajan, and Nikolaos Papanikolaou. "QMC: a model checker for quantum systems". In: *Proceeding of the 20th International Conference on Computer Aided Verification (CAV 2008)* (Princeton, NJ, USA). Ed. by Aarti Gupta and Sharad Malik. Vol. 5123. Lecture Notes in Computer Science. Springer, 2008, pp. 543–547. ISBN: 978-3-540-70543-7. DOI: 10.1007/978-3-540-70545-1_51.

[Gol70] Donald Goldfarb. "A family of variable-metric methods derived by variational means". In: *Mathematics of Computation* 24 (1970), pp. 23–26.

[Gra88] Steven K. Graham. "Closure properties of a probabilistic domain construction". In: *Proceedings of the 3rd Workshop on Mathematical Foundations of Programming Language Semantics, MFPS'87* (Tulane University, New Orleans, Louisiana, USA). Ed. by Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt. Vol. 298. Lecture Notes in Computer Science. Springer, 1988, pp. 213–233. ISBN: 3-540-19020-1. DOI: 10.1007/3-540-19020-1.

[Gro96] Lov K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, STOC'96* (Philadelphia, Pennsylvania, USA, May 22–June 24, 1996). Ed. by Gary L. Miller. ACM, 1996, pp. 212–219. ISBN: 0-89791-785-5. DOI: 10.1145/237814.237866.

[GS10] Dan R. Ghica and Alex I. Smith. "Geometry of synthesis II: from games to delay-insensitive circuits". In: *Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2010, Ottawa, Ontario, Canada, May 6-10, 2010* (Ottawa, Ontario, Canada). Ed. by Michael W. Mislove and Peter Selinger. Vol. 265. Electronic Notes in Theoretical Computer Science. Elsevier, May 2010, pp. 301–324. DOI: 10.1016/j.entcs.2010.08.018.

[GS11] Dan R. Ghica and Alex I. Smith. "Geometry of synthesis III: resource management through type inference". In: *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011* (Austin, TX, USA). Ed. by Thomas Ball and Mooly Sagiv. ACM, Jan. 2011, pp. 345–356. ISBN: 978-1-4503-0490-0. DOI: 10.1145/1926385.1926425.

[GSS11] Dan R. Ghica, Alex Smith, and Satnam Singh. "Geometry of synthesis IV". In: [ICFP11], pp. 221–233. DOI: 10.1145/2034773.2034805]].

[GSS92] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. "Bounded linear logic: a modular approach to polynomial-time computability". In: *Theoretical Computer Science* 97.1 (1992), pp. 1–66. DOI: 10.1016/0304-3975(92)90386-T.

[Had15] Amar Hadzihasanovic. "A diagrammatic axiomatisation for qubit entanglement". In: [LICS15]. DOI: 10.1109/LICS.2015.59.

[Had17] Amar Hadzihasanovic. "The algebra of entanglement and the geometry of composition". PhD thesis. Oxford Univesity, 2017.

[HAGH16] Marieke Huisman, Wolfgang Ahrendt, Daniel Grahl, and Martin Hentschel. "Formal specification with the java modeling language". In: *Deductive Software Verification - The KeY Book - From Theory to Practice*. Ed. by Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, Peter H. Schmitt, and Mattias Ulbrich. Vol. 10001. Lecture Notes in Computer Science. Springer, 2016. Chap. 7, pp. 193–241. ISBN: 978-3-319-49811-9. DOI: 10.1007/978-3-319-49812-6_7.

[Hal07] Sean Hallgren. "Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem". In: *Journal of the ACM* 54.1 (2007), 4:1–4:19. DOI: 10.1145/1206035.1206039.

[Hal92] J.S. Hall. "An electroid switching model for reversible computer architectures". In: *Workshop on Physics and Computation*. 1992, pp. 237–247. DOI: 10.1109/PHYCMP.1992.615549.

[Har87] Rogers Hartley Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987. ISBN: 978-0-262-68052-3.

[HC18] Luke E Heyfron and Earl T Campbell. "An efficient quantum compiler that reduces T count". In: *Quantum Science and Technology* 4.1 (Sept. 2018), p. 015004. DOI: 10.1088/2058-9565/aad604.

[HD96] John Hatcliff and Olivier Danvy. *Thunks and the Lambda-Calculus*. Tech. rep. RS-96-19. BRICS, University of Aahrus, 1996.

[HFGB+23] Nicolas Heurtel, Andreas Fyrillas, Grégoire de Gliniasty, Raphaël Le Bihan, Sébastien Malherbe, Marceau Pailhas, Eric Bertasi, Boris Bourdoncle, Pierre-Emmanuel Emeriau, Rawad Mezher, Luka Music, Nadia Belabas, Benoît Valiron, Pascale Senellart, Shane Mansfield, and Jean Senellart. "Perceval: a software platform for discrete variable photonic quantum computing". In: *Quantum* 7 (2023), p. 931. DOI: 10.22331/Q-2023-02-21-931. HAL: hal-03874624.

[HH11] Ichiro Hasuo and Naohiko Hoshino. "Semantics of higher-order quantum computation via geometry of interaction". In: [LICS11], pp. 237–246. DOI: 10.1109/LICS.2011.26.

[HH17] Ichiro Hasuo and Naohiko Hoshino. "Semantics of higher-order quantum computation via geometry of interaction". In: *Annals of Pure and Applied Logic* 168.2 (2017). Long version of LICS paper [HH11]., pp. 404–469. DOI: 10.1016/j.apal.2016.10.010.

[HH19] Reiner Hähnle and Marieke Huisman. "Deductive software verification: from pen-and-paper proofs to industrial tools". In: *Computing and Software Science - State of the Art and Perspectives*. Ed. by Bernhard Steffen and Gerhard J. Woeginger. Vol. 10000. Lecture Notes in Computer Science. Springer, 2019, pp. 345–373. DOI: 10.1007/978-3-319-91908-9_18.

[HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum algorithm for linear systems of equations". In: *Physical Review Letters* 103 (15 2009), p. 150502. DOI: 10.1103/PhysRevLett.103.150502. arXiv: 0811.3171.

[HHZYHW19] Shih-Han Hung, Kesha Hietala, Shaopeng Zhu, Mingsheng Ying, Michael Hicks, and Xiaodi Wu. "Quantitative robustness analysis of quantum programs". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), 31:1–31:29. DOI: 10.1145/3290344.

[HK15] Chris Heunen and Martti Karvonen. "Reversible monadic computing". In: *Proceedings of the 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS XXXI* (Nijmegen, The Netherlands). Ed. by Dan Ghica. Vol. 319. Electronic Notes in Theoretical Computer Science. 2015, pp. 217–237. DOI: 10.1016/j.entcs.2015.12.014. arXiv: 1505.04330.

[HKK18] Chris Heunen, Robin Kaarsgaard, and Martti Karvonen. "Reversible effects as inverse arrows". In: *Proceedings of the 34th Conference on the Mathematical Foundations of Programming Semantics, MFPS XXXIV* (Dalhousie University, Halifax, Canada). Ed. by Sam Staton. Vol. 341. Electronic Notes in Theoretical Computer Science. Elsevier, 2018, pp. 179–199. DOI: 10.1016/j.entcs.2018.11.009.

[HM19a] Yipeng Huang and Margaret Martonosi. "QDB: from quantum algorithms towards correct quantum programs". In: *Proceedings of the 9th Workshop on Evaluation and Usability of Programming Languages and Tools, PLATEAU@SPLASH 2018* (Boston, Massachusetts, USA, Nov. 5, 2018). Ed. by Titus Barik, Joshua Sunshine, and Sarah E. Chasins. Vol. 67. OASICS. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 4:1–4:14. ISBN: 978-3-95977-091-0. DOI: 10.4230/OASIcs.PLATEAU.2018.4. URL: http://www.dagstuhl.de/dagpub/978-3-95977-091-0.

[HM19b] Yipeng Huang and Margaret Martonosi. "Statistical assertions for validating patterns and finding bugs in quantum programs". In: *Proceedings of the 46th International Symposium on Computer Architecture, ISCA 2019* (Phoenix, AZ, USA, June 22–26, 2019). Ed. by Srilatha Bobbie Manne, Hillery C. Hunter, and Erik R. Altman. ACM, 2019, pp. 541–553. ISBN: 978-1-4503-6669-4. DOI: 10.1145/3307650.3322213.

[HMH14] Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. "Memoryful geometry of interaction: from coalgebraic components to algebraic effects". In: [LICS14], 52:1–52:10. DOI: 10.1145/2603088.2603124.

[HMSM18] Winston Haaswijk, Alan Mishchenko, Mathias Soeken, and Giovanni De Micheli. "SAT based exact synthesis using DAG topology families". In: *Proceedings of the 55th Annual Design Automation Conference, DAC 2018* (San Francisco, CA, USA). ACM, 2018, 53:1–53:6. ISBN: 978-1-5386-4114-9. DOI: 10.1145/3195970.3196111.

[HMSV23] Nicolas Heurtel, Shane Mansfield, Jean Senellart, and Benoît Valiron. "Strong simulation of linear optical processes". In: *Computer Physics Communications* 291 (2023), p. 108848. DOI: 10.1016/J.CPC.2023.108848. HAL: hal-03874624v1.

[HNW18] Amar Hadzihasanovic, Kang Feng Ng, and Quanlong Wang. "Two complete axiomatisations of pure-state qubit quantum computing". In: [LICS18], pp. 502–511. DOI: 10.1145/3209108.3209128.

[Hoa69] C. A. R. Hoare. "An axiomatic basis for computer programming". In: *Communications of the ACM* 12.10 (1969), pp. 576–580. DOI: 10.1145/363235.363259.

[Hor11] Dominic Horsman. "Quantum picturalism for topological cluster-state computing". In: *New Journal of Physics* 13.9 (Sept. 2011), p. 095011. DOI: 10.1088/1367-2630/13/9/095011. arXiv: 1101.4722.

[How80] W. A. Howard. "The formulae-as-type notion of construction". In: *To H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*. Ed. by Jonathan Paul Seldin and James Roger Hindley. Academic Press, 1980.

[HRHLH21] Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. "Proving quantum programs correct". In: *Proceedings of the 12th International Conference on Interactive Theorem Proving, ITP 2021* (Rome, Italy (Virtual Conference), June 29–July 1, 2021). Ed. by Liron Cohen and Cezary Kaliszyk. Vol. 193. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 21:1–21:19. ISBN: 978-3-95977-188-7. DOI: 10.4230/LIPIcs.ITP.2021.21.

[HRHWH21] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. "A verified optimizer for quantum circuits". In: *Proceedings of the ACM on Programming Languages* 5.POPL (2021), 37:1–37:29. DOI: 10.1145/3434318.

[HRS18] Thomas Häner, Martin Roetteler, and Krysta M. Svore. "Optimizing Quantum Circuits for Arithmetic". Draft. 2018. arXiv: 1805.12445.

[HSRS18] Thomas Häner, Mathias Soeken, Martin Roetteler, and Krysta M. Svore. "Quantum circuits for floating-point arithmetic". In: [RC18], pp. 162–174. DOI: 10.1007/978-3-319-99498-7_11.

[IC1405] *Official Webpage of the COST Action IC1405 on Reversible Computation*. URL: http://www.revcomp.eu/ (visited on Aug. 27, 2021).

[ICFP11] Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy, eds. *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011* (Tokyo, Japan, Sept. 19–21, 2011). ACM, 2011. ISBN: 978-1-4503-0865-6.

[Ing61] Peter Z. Ingerman. "Thunks: a way of compiling procedure statements with some comments on procedure declarations". In: *Communications of the ACM* 4.1 (1961), pp. 55–58.

[JKT18] Petur Andrias Højgaard Jacobsen, Robin Kaarsgaard, and Michael Kirkedal Thomsen. "CoreFun : a typed functional reversible core language". In: [RC18], pp. 304–321. DOI: 10.1007/978-3-319-99498-7_21.

[Jon87] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice Hall, 1987.

[Jon90] Claire Jones. "Probabilistic Non-Determinism". PhD thesis. University of Edinburgh, 1990.

[JPKH+15] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. "ScaffCC: scalable compilation and analysis of quantum programs". In: *Parallel Computing* 45 (2015), pp. 2–17. DOI: 10.1016/j.parco.2014.12.001.

[JPV18] Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. "A complete axiomatisation of the ZX-calculus for Clifford+T quantum mechanics". In: [LICS18], pp. 559–568. DOI: 10.1145/3209108.3209131. arXiv: 1705.11151.

[Jr08] Frederick P. Brooks Jr. *The Mythical Man-Month - Essays on Software Engineering*. Anniversary Edition. Addison-Wesley, 2008. ISBN: 0-201-83595-9.

[JS12a] Roshan P. James and Amr Sabry. *Embracing the Laws of Physics*. Presented at OBT'12. 2012.

[JS12b] Roshan P. James and Amr Sabry. "Information effects". In: *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'12* (Philadelphia, Pennsylvania, USA). Ed. by John Field and Michael Hicks. ACM, 2012, pp. 73–84. ISBN: 978-1-4503-1083-3. DOI: 10.1145/2103656.2103667. URL: http://dl.acm.org/citation.cfm?id=2103656.

[JS12c] Roshan P. James and Amr Sabry. "Isomorphic interpreters from logically reversible abstract machines". In: *Post-Proceedings of the 4th International Workshop on Reversible Computation, RC'12* (Copenhagen, Denmark). Ed. by Robert Glück and Tetsuo Yokoyama. Vol. 7581. Lecture Notes in Computer Science. Springer, 2012, pp. 57–71. DOI: 10.1007/978-3-642-36315-3_5.

[JS14] Rosham P. James and Amr Sabry. "Theseus: A High-Level Language for Reversible Computing". Booklet of work-in-progress and short reports for RC 2014. 2014.

[JT98] Achim Jung and Regina Tix. "The troublesome probabilistic powerdomain". In: *Comprox III, Third Workshop on Computation and Approximation* (Birmingham, England, Sept. 11–13, 1997). Ed. by Abbas Edalat, Achim Jung, Klaus Keimel, and Marta Kwiatkowska. Vol. 13. Electronic Notes in Theoretical Computer Science. 1998, pp. 70–91. DOI: 10.1016/S1571-0661(05)80216-6.

[Kaa19] Robin Kaarsgaard. "Inversion, iteration, and the art of dual wielding". In: [RC19], pp. 34–50. DOI: 10.1007/978-3-030-21500-2_3. arXiv: 1904.01679.

[KAG17] Robin Kaarsgaard, Holger Bock Axelsen, and Robert Glück. "Join inverse categories and reversible recursion". In: *Journal of Logical and Algebraic Methods in Programming* 87 (2017), pp. 33–50. ISSN: 2352-2208. DOI: 10.1016/j.jlamp.2016.08.003.

[Kas79] J. Kastl. "Inverse categories". In: *Algebraische Modelle, Kategorien und Gruppoide*. Studien zur Algebra und ihre Anwendungen, Band 7. Berlin, Akademie-Verlag, 1979, pp. 51–60.

[KIQBAW19] Nathan Killoran, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. "Strawberry Fields: a software platform for photonic quantum computing". In: *Quantum* 3 (2019), p. 129. DOI: 10.22331/q-2019-03-11-129. arXiv: 1804.03159.

[Kit95] A Yu Kitaev. "Quantum measurements and the Abelian stabilizer problem". 1995. arXiv: quant-ph/9511026.

[KJS10] Oleg Kiselyov, Simon Peyton Jones, and Chung-Chieh Shan. "Fun with type functions". In: *Reflections on the Work of C.A.R. Hoare*. Springer, 2010. Chap. 14, pp. 301–331. DOI: 10.1007/978-1-84882-912-1_14.

[KKPSY15] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. "Frama-C: a software analysis perspective". In: *Formal Aspects of Computing* 27.3 (2015), pp. 573–609. DOI: 10.1007/s00165-014-0326-7.

[Kle35a] Stephen C. Kleene. "A theory of positive integers in formal logic, part I". In: *American Journal of Mathematics* 57.1 (1935), pp. 153–173. DOI: 10.2307/2372027.

[Kle35b] Stephen C. Kleene. "A theory of positive integers in formal logic, part II". In: *American Journal of Mathematics* 57.2 (1935), pp. 219–244. DOI: 10.2307/2371199.

[Kle45] Stephen Cole Kleene. "On the interpretation of intuitionistic number theory". In: *Journal of Symbolic Logic* 10.4 (1945), pp. 109–124. DOI: 10.2307/2269016.

[Kle67] Stephen Cole Kleene. *Mathematical Logic*. Dover Publication Inc., 1967.

[Klu99] Werner E. Kluge. "A reversible SE(M)CD machine". In: *Selected Papers of the 11th International Workshop on the Implementation of Functional Languages, IFL'99* (Lochem, The Netherlands, Sept. 7–10, 1999). Ed. by Pieter W. M. Koopman and Chris Clack. Vol. 1868. Lecture Notes in Computer Science. Springer, 1999, pp. 95–113. ISBN: 3-540-67864-6. DOI: 10.1007/10722298_6.

[KMR89] Jan Willem Klop, J. J. C. Meijer, and Jan J. M. M. Rutten, eds. *J.W. de Bakker, 25 Jaar Semantiek: Liber Amicorum*. CWI, 1989. URL: https://ir.cwi.nl/pub/20371/ (visited on Aug. 11, 2022).

[Kni95] E. Knill. *Approximation by Quantum Circuits*. Tech. rep. LANL report LAUR-95-2225. Los Alamos National Laboratory, 1995.

[Kni96] Emanuel H. Knill. *Conventions for Quantum Pseudocode*. Tech. rep. LAUR-96-2724. Los Alamos, New Mexico, US.: Los Alamos National Laboratory, 1996.

[Koz83] Dexter Kozen. "Results on the propositional μ-calculus". In: *Theoretical Computer Science* 27 (1983), pp. 333–354. DOI: 10.1016/0304-3975(82)90125-6.

[KR21] Robin Kaarsgaard and Mathys Rennela. "Join inverse rig categories for reversible functional programming, and beyond". In: [MFPS21], pp. 152–167. DOI: 10.4204/EPTCS.351.10.

[Kri07] Jean-Louis Krivine. "A call-by-name lambda-calculus machine". In: *Higher-Order and Symbolic Computation* 20.3 (2007), pp. 199–207. DOI: 10.1007/s10990-007-9018-9.

[KV19] Robin Kaarsgaard and Niccolò Veltri. "En garde! unguarded iteration for reversible computation in the delay monad". In: *Proceedings of the 13th International Conference on Mathematics of Program Construction, MPC 2019* (Porto, Portugal). Ed. by Graham Hutton. Vol. 11825. Lecture Notes in Computer Science. Springer Verlag, Oct. 2019, pp. 366–384. ISBN: 978-3-030-33635-6. DOI: 10.1007/978-3-030-33636-3_13.

[KW20] Aleks Kissinger and John van de Wetering. "Reducing the number of non-Clifford gates in quantum circuits". In: *Physical Review A* 102 (2 2020), p. 022406. DOI: 10.1103/PhysRevA.102.022406. arXiv: 1903.10477.

[KZ15] Aleks Kissinger and Vladimir Zamdzhiev. "Quantomatic: a proof assistant for diagrammatic reasoning". In: *Proceedings of the 25th International Conference on Automated Deduction, CADE-25* (Berlin, Germany, Aug. 1–7, 2015). Ed. by Amy P. Felty and Aart Middeldorp. Vol. 9195. Lecture Notes in Computer Science. Springer, 2015, pp. 326–336. ISBN: 978-3-319-21400-9. DOI: 10.1007/978-3-319-21401-6_22. arXiv: 1503.01034.

[Laf04] Yves Lafont. "Soft linear logic and polynomial time". In: *Theoretical Computer Science* 318.1–2 (2004), pp. 163–180. DOI: 10.1016/j.tcs.2003.10.018.

[Laf90] Yves Lafont. "Interaction nets". In: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'90* (San Francisco, California, USA). Ed. by Frances E. Allen. ACM, 1990, pp. 95–108. ISBN: 0-89791-343-4. DOI: 10.1145/96709.96718.

[Laf95] Yves Lafont. "From proof-nets to interaction nets". In: [GLR95], pp. 225–247.

[Lan02] Serge Lang. *Algebra*. Revised Third Edition. Vol. 211. Graduate Texts in Mathematics. Springer, 2002. ISBN: 0-387-95385-X.

[Lan61] Rolf Landauer. "Irreversibility and heat generation in the computing process". In: *IBM Journal of Research and Development* 5.3 (1961), pp. 183–191. DOI: 10.1147/rd.53.0183.

[Lan66] Peter J. Landin. "The next 700 programming languages". In: *Communications of the ACM* 9.3 (1966), pp. 157–166. DOI: 10.1145/365230.365257. URL: https://doi.org/10.1145/365230.365257.

[Lau13] Olivier Laurent. "An Introduction to Proof-Nets". Notes. 2013.

[LB21] Robin Lorenz and Jonathan Barrett. "Causal and compositional structure of unitary transformations". In: *Quantum* 5 (2021), p. 511. DOI: 10.22331/q-2021-07-28-511. arXiv: 2001.07774.

[LBK05] Yuan Liang Lim, Almut Beige, and Leong Chuan Kwek. "Repeat-Until-Success linear optics distributed quantum computing". In: *Physical Review Letters* 95 (3 2005), p. 030505. DOI: 10.1103/PhysRevLett.95.030505.

[Lee22] Dongho Lee. "Formal Methods for Quantum Programming Languages". Thèse de doctorat. Université Paris-Saclay, 2022.

[Lee90] Jan van Leeuwen, ed. *Formal Models and Semantics*. Vol. B. Handbook of Theoretical Computer Science. Elsevier, 1990. ISBN: 0-444-88074-7.

[Lem24] Louis Lemonnier. "The Semantics of Effects : Centrality, Quantum Control and Reversible Recursion". PhD thesis. Université Paris Saclay, 2024.

[Lep16] Rodolphe Lepigre. "Semantics and Implementation of an Extension of ML for Proving Programs". Thèse de Doctorat. Université de Grenoble, 2016.

[Lev16] Thomas Leventis. "Probabilistic λ-Theories". Thèse de Doctorat. Aix-Marseille Université, 2016. HAL: tel-01427279.

[LFFP11] Michael Leuschel, Jérôme Falampin, Fabian Fritz, and Daniel Plagge. "Automated property verification for large scale B models with ProB". In: *Formal Aspects of Computing* 23.6 (2011), pp. 683–709. DOI: 10.1007/s00165-010-0172-1.

[LFHY14] Ugo Dal Lago, Claudia Faggian, Ichiro Hasuo, and Akira Yoshimizu. "The geometry of synchronization". In: [LICS14], 35:1–35:10. DOI: 10.1145/2603088.2603154. arXiv: 1405.3427.

[LFSMC20] Andrew Litteken, Yung-Ching Fan, Devina Singh, Margaret Martonosi, and Frederic T Chong. "An updated LLVM-based quantum research compiler with further OpenQASM support". In: *Quantum Science and Technology* 5.3 (2020), p. 034013.

[LFVY15] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. "Parallelism and synchronization in an infinitary context". In: [LICS15], pp. 559–572. DOI: 10.1109/LICS.2015.58. HAL: hal-01231831. arXiv: 1505.03635.

[LFVY17] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. "The geometry of parallelism: classical, probabilistic, and quantum effects". In: [POPL17], pp. 833–845. DOI: 10.1145/3009837.3009859. HAL: hal-01474620. arXiv: 1610.09629.

[LH09] Ugo Dal Lago and Martin Hofmann. "Bounded linear logic, revisited". In: *Proceedings of the 9th International Conference on Typed Lambda Calculi and Applications (TLCA'09)* (Brasilia, Brazil). Ed. by Pierre-Louis Curien. Vol. 5608. Lecture Notes in Computer Science. See also the journal's version [LH10]. Springer, 2009. ISBN: 978-3-642-02272-2. DOI: 10.1007/978-3-642-02273-9_8.

[LH10] Ugo Dal Lago and Martin Hofmann. "Bounded linear logic, revisited". In: *Logical Methods in Computer Science* 6.4 (2010). Long version of the TLCA'09 publication [LH09]. DOI: 10.2168/LMCS-6(4:7)2010. arXiv: 0904.2675.

[LH85] David C. Luckham and Friedrich W. von Henke. "An overview of Anna, a specification language for Ada". In: *IEEE Software* 2.2 (1985), pp. 9–22. DOI: 10.1109/MS.1985.230345.

[LICS04] *Proceedings of the 19th Symposium on Logic in Computer Science, LICS'04* (Turku, Finland, July 14–17, 2004). IEEE. IEEE Computer Society Press, July 2004. ISBN: 0-7695-2192-4.

[LICS11] *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011* (Toronto, Ontario, Canada, June 21–24, 2011). IEEE Computer Society, 2011. ISBN: 978-0-7695-4412-0.

[LICS14] Thomas A. Henzinger and Dale Miller, eds. *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science, CSL-LICS'14* (Vienna, Austria). ACM, 2014. ISBN: 978-1-4503-2886-9.

[LICS15] *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'15* (Kyoto, Japan). IEEE Computer Society, 2015. ISBN: 978-1-4799-8875-4.

[LICS18] Anuj Dawar and Erich Grädel, eds. *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'2018* (Oxford, UK). ACM, 2018. DOI: 10.1145/3209108.

[LICS19] *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'19* (Vancouver, BC, Canada, June 24–27, 2019). IEEE, 2019. ISBN: 978-1-7281-3608-0.

[LMMP13] Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. "Weighted relational models of typed lambda-calculi". In: *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013* (New Orleans, LA, USA, June 25–28, 2013). IEEE Computer Society, 2013, pp. 301–310. ISBN: 978-1-4799-0413-6. DOI: 10.1109/LICS.2013.36.

[LMZ10] Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. "Quantum implicit computational complexity". In: *Theoretical Computer Science* 411.2 (2010), pp. 377–409. DOI: 10.1016/j.tcs.2009.07.045.

[LMZ18] Bert Lindenhovius, Michael W. Mislove, and Vladimir Zamdzhiev. "Enriching a linear/non-linear lambda calculus: A programming language for string diagrams". In: [LICS18], pp. 659–668. DOI: 10.1145/3209108.3209196. HAL: hal-03018477. arXiv: 1804.09822.

[LN98] K. Rustan M. Leino and Greg Nelson. "An extended static checker for Modular-3". In: *Proceedings of the 7th International Conference on Compiler Construction, CC'98* (Lisbon, Portugal, Mar. 28–Apr. 4, 1998). Ed. by Kai Koskimies. Vol. 1383. Lecture Notes in Computer Science. Springer, 1998, pp. 302–305. ISBN: 3-540-64304-4. DOI: 10.1007/BFb0026441.

[Loe50] Charles Loewner. "Some classes of functions defined by difference or differential inequalities". In: *Bulletin of the American Mathematical Society* 56.6 (1950), pp. 308–319.

[Low19] Guang Hao Low. "Hamiltonian simulation with nearly optimal dependence on spectral norm". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019* (Phoenix, AZ, USA, June 23–26, 2019). Ed. by Moses Charikar and Edith Cohen. ACM, 2019, pp. 491–502. DOI: 10.1145/3313276.3316386. arXiv: 1807.03967.

[Löw34] Karl Löwner. "Über monotone Matrixfunktionen". In: *Mathematische Zeitschrift* 38.1 (1934), pp. 177–216. DOI: 10.1007/BF01170633.

[LPVX21] Dongho Lee, Valentin Perrelle, Benoît Valiron, and Zhaowei Xu. "Concrete categorical model of a quantum circuit description language with measurement". In: *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021*. Ed. by Mikolaj Bojanczyk and Chandra Chekuri. Vol. 213. LIPIcs. 2021, 51:1–51:20. DOI: 10.4230/LIPIcs.FSTTCS.2021.51.

[LS89] Joachim Lambek and Philip Scott. *Introduction to Higher Order Categorical Logic*. 2nd ed. Vol. 7. Cambridge studies in advanced mathematics. Cambridge University Press, 1989. ISBN: 0-521-35653-9.

[LS90] Robert Y. Levin and Alan T. Sherman. "A note on Bennett's time-space tradeoff for reversible computation". In: *SIAM Journal on Computing* 19.4 (1990), pp. 673–677. DOI: 10.1137/0219046.

[Lut86] Christopher Lutz. "Janus: a time-reversible language". Letter to R. Landauer, posted online by Tetsuo Yokoyama on http://www.tetsuo.jp/ref/janus.html. 1986.

[LWZG+18] Shusen Liu, Xin Wang, Li Zhou, Ji Guan, Yinan Li, Yang He, Runyao Duan, and Mingsheng Ying. "Q|SI⟩ : a quantum programming environment". In: *Symposium on Real-Time and Hybrid Systems - Essays Dedicated to Professor Chaochen Zhou on the Occasion of His 80th Birthday*. Ed. by Cliff B. Jones, Ji Wang, and Naijun Zhan. Vol. 11180. Lecture Notes in Computer Science. 2018, pp. 133–164. DOI: 10.1007/978-3-030-01461-2_8.

[LY18] Yangjia Li and Mingsheng Ying. "Algorithmic analysis of termination problems for quantum programs". In: *Proceedings of the ACM on Programming Languages* 2.POPL (2018), 35:1–35:29. DOI: 10.1145/3158123.

[LYY14] Yangjia Li, Nengkun Yu, and Mingsheng Ying. "Termination of nondeterministic quantum programs". In: *Acta Informatica* 51.1 (2014), pp. 1–24. DOI: 10.1007/s00236-013-0185-3.

[LZ12] Ugo Dal Lago and Margherita Zorzi. "Probabilistic operational semantics for the lambda calculus". In: *RAIRO Theor. Informatics Appl.* 46.3 (2012), pp. 413–450. DOI: 10.1051/ita/2012012. arXiv: 1104.0195.

[LZ14] Ugo Dal Lago and Margherita Zorzi. "Wave-style token machines and quantum lambda calculi". In: *Proceedings Third International Workshop on Linearity, LINEARITY 2014* (Vienna, Austria, July 13, 2014). Ed. by Sandra Alves and Iliano Cervesato. Vol. 176. Electronic Proceedings in Theoretical Computer Science. 2014, pp. 64–78. DOI: 10.4204/EPTCS.176.6.

[LZBY22] Junyi Liu, Li Zhou, Gilles Barthe, and Mingsheng Ying. "Quantum weakest preconditions for reasoning about expected runtimes of quantum programs". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '22* (Haifa, Israel, Aug. 2–5, 2022). Ed. by Christel Baier and Dana Fisman. ACM, 2022, 4:1–4:13. ISBN: 978-1-4503-9351-5. DOI: 10.1145/3531130.3533327.

[LZWY+19a] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. "Formal verification of quantum algorithms using quantum Hoare logic". In: *Proceedings of the 31st International Conference on Computer Aided Verification, CAV 2019, Part II* (New York City, NY, USA, July 15–18, 2019). Ed. by Isil Dillig and Serdar Tasiran. Vol. 11562. Lecture Notes in Computer Science. Springer, 2019, pp. 187–207. ISBN: 978-3-030-25542-8. DOI: 10.1007/978-3-030-25543-5_12. arXiv: 1601.03835.

[LZWY+19b] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. "Quantum Hoare logic". In: *Archive of Formal Proofs* 2019 (2019). URL: https://www.isa-afp.org/entries/QHLProver.html (visited on Aug. 15, 2022).

[LZYDYX20] Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. "Projection-based runtime assertions for testing and debugging quantum programs". In: *Proceedings of the ACM on Programming Languages* 4.OOPSLA (2020), 150:1–150:29. DOI: 10.1145/3428218. arXiv: 1911.12855.

[Mac94] Ian Mackie. "The Geometry of Implementation : Applications of the Geometry of Interaction to Language Implementation". PhD thesis. University of London, 1994.

[Mac95] Ian Mackie. "The geometry of interaction machine". In: *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'95* (San Francisco, California, US.). ACM. ACM Press, 1995, pp. 198–208. DOI: 10.1145/199448.199483.

[Mal10] Octavio Malherbe. "Categorical models of computation: partially traced categories and presheaf models of quantum computation". PhD thesis. University of Ottawa, 2010. arXiv: 1301.5087.

[Mar71] Per Martin-Löf. "Hauptsatz for the intuitionistic theory of iterated inductive definitions". In: *Proceedings of the Second Scandinavian Logic Symposium*. Ed. by J. E. Fenstad. Vol. 63. Studies in Logic and the Foundations of Mathematics. North-Holland, 1971, pp. 179–216. DOI: 10.1016/S0049-237X(08)70847-4.

[Mar84] Per Martin-Löf. *Intuitionistic Type Theory*. Studies in Proof Theories. Napoli, Italy: Bibliopolis, 1984.

[Mat03] Armando B. Matos. "Linear programs in a simple reversible language". In: *Theoretival Computer Science* 290.3 (2003), pp. 2063–2074. DOI: 10.1016/S0304-3975(02)00486-3.

[MAT14] Mohamed Yousri Mahmoud, Vincent Aravantinos, and Sofiène Tahar. "Formal verification of optical quantum flip gate". In: *Proceedings of the 5th International Conference on Interactive Theorem Proving, ITP'14* (Vienna, Austria). Ed. by Gerwin Klein and Ruben Gamboa. Vol. 8558. Lecture Notes in Computer Science. Springer, 2014, pp. 358–373. DOI: 10.1007/978-3-319-08970-6_23.

[Mat98] Ralph Matthes. "Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types". PhD thesis. Ludwig-Maximilians-Universität, München, Germany, 1998.

[Maz06] Damiano Mazza. "Interaction Nets : Semantics and Concurrent Extensions". Thèse de Doctorat. Université de la Méditerranée/Università degli Studi Roma Tre, 2006.

[McKinsey21] Matteo Biondi, Anna Heid, Ivan Ostojic, Nicolaus Henke, Lorenzo Pautasso, Niko Mohr, Linde Wester, and Rodney Zemmel. *Quantum computing: An emerging ecosystem and industry use cases*. Report. McKinsey & Company, 2021.

[MDM03] Dmitri Maslov, Gerhard W. Dueck, and D. Michael Miller. "Fredkin/Toffoli templates for reversible logic synthesis". In: *Proceedings of the International Conference on Computer-Aided Design, ICCAD'03* (San Jose, CA, USA). IEEE Computer Society / ACM, 2003, pp. 256–261. ISBN: 1-58113-762-1. DOI: 10.1109/ICCAD.2003.1257667.

[MDM05] Dmitri Maslov, Gerhard W. Dueck, and D. Michael Miller. "Toffoli network synthesis with templates". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.6 (2005), pp. 807–817. DOI: 10.1109/TCAD.2005.847911.

[Mei24] Arianne Meijer–van de Griend. "Advances in Quantum Compilation in the NISQ Era". PhD thesis. University of Helsinki, 2024.

[Men88] Paul Mendler. "Inductive Definition in Type Theory". PhD thesis. Cornell University, USA, 1988. URL: https://hdl.handle.net/1813/6710 (visited on Aug. 3, 2022).

[Mey92] Bertrand Meyer. "Applying "design by contract"". In: *IEEE Computer* 25.10 (1992), pp. 40–51. DOI: 10.1109/2.161279.

[MFPS21] Ana Sokolova, ed. *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics (MFPS 2021)* (Hybrid: Salzburg, Austria and Online, Aug. 30–Sept. 2, 2021). Vol. 351. EPTCS. 2021. DOI: 10.4204/EPTCS.351.

[MFPS93] Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, eds. *Mathematical Foundations of Programming Semantics: Ninth International Conference, MFPS IX* (New Orleans, Louisiana, US.). Vol. 802. Lecture Notes in Computer Science. Springer Verlag, Apr. 1993. ISBN: 3-540-58027-1. DOI: 10.1007/3-540-58027-1.

[MHT04] Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. "An injective language for reversible computation". In: *Proceedings of the 7th International Conference on Mathematics of Program Construction, MPC 2004* (Stirling, Scotland, UK). Ed. by Dexter Kozen. Vol. 3125. Lecture Notes in Computer Science. Springer Verlag, July 2004, pp. 289–313. ISBN: 978-3-540-22380-1. DOI: 10.1007/978-3-540-27764-4_16.

[Mit78] Peter Mittelstaedt. *Quantum Logic*. Vol. 126. Synthese Library. Dordrecht, Holland: D. Reidel Publishing Company, 1978. ISBN: 978-94-009-9873-5. DOI: 10.1007/978-94-009-9871-1.

[MM16] Olivia Di Matteo and Michele Mosca. "Parallelizing quantum circuit synthesis". In: *Quantum Science and Technology* 1.1 (Mar. 2016), p. 015003. DOI: 10.1088/2058-9565/1/1/015003.

[MM90] Per Martin-Löf and Grigori Mints, eds. *Proceedings of the International Conference on Computer Logic (COLOG-88)* (Tallinn, USSR). Vol. 417. Lecture Notes in Computer Science. Springer, 1990.

[MMNSB16] Esteban A. Martinez, Thomas Monz, Daniel Nigg, Philipp Schindler, and Rainer Blatt. "Compiling quantum algorithms for architectures with multi-qubit gates". In: *New Journal of Physics* 18 (2016), p. 063029.

[Mog14] Torben Ægidius Mogensen. "Reference counting for reversible languages". In: *Proceedings of hte 6th International Conference on Reversible Computation, RC 2014* (Kyoto, Japan, July 10–11, 2014). Ed. by Shigeru Yamashita and Shin-ichi Minato. Vol. 8507. Lecture Notes in Computer Science. Springer, 2014, pp. 82–94. ISBN: 978-3-319-08493-0. DOI: 10.1007/978-3-319-08494-7_7.

[Mog18] Torben Ægidius Mogensen. "Reversible garbage collection for reversible functional languages". In: *New Generation Computing* 36.3 (2018), pp. 203–232. DOI: 10.1007/s00354-018-0037-3.

[Mog19] Torben Ægidius Mogensen. "Reversible in-place carry-lookahead addition with few ancillae". In: [RC19], pp. 224–237. DOI: 10.1007/978-3-030-21500-2_14.

[Mog89] Eugenio Moggi. "Computational lambda-calculus and monads". In: *Proceedings of the Fourth Symposium on Logic in Computer Science, LICS'89* (Pacific Grove, California, US.). IEEE. IEEE Computer Society Press, June 1989, pp. 14–23. ISBN: 0-8186-1954-6. DOI: 10.1109/LICS.1989.39155. URL: http://www.lfcs.inf.ed.ac.uk/reports/88/ECS-LFCS-88-66/.

[Moo62] Edward F. Moore. "Machine models of self-reproduction". In: *Mathematics Problems in Biological Sciences* (New York City, US. Apr. 5–8, 1961). Ed. by R. E. Bellman. Vol. XIV. Proceedings of a Symposia in Applied Mathematics. AMS, 1962, pp. 17–33.

[MOTW95] John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. "Call-by-name, call-by-value, call-by-need and the linear lambda calculus". In: *Proceedings of the 11th Annual Conference on Mathematical Foundations of Programming Semantics, MFPS XI* (New Orleans, Louisiana, USA). Vol. 1. Electronic Notes in Theoretical Computer Science. See also journal version [MOTW99]. 1995, pp. 370–392. DOI: 10.1016/S1571-0661(04)00022-2.

[MOTW99] John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. "Call-by-name, call-by-value, call-by-need and the linear lambda calculus". In: *Theoretical Computer Science* 228.1-2 (1999). Long version of [MOTW95]., pp. 175–210. DOI: 10.1016/S0304-3975(98)00358-2.

[MRBA16] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. "The theory of variational hybrid quantum-classical algorithms". In: *New Journal of Physics* 18 (2016), p. 023023. DOI: 10.1088/1367-2630/18/2/023023. arXiv: 1509.04279.

[MS99] Klaus Mølmer and Anders Sørensen. "Multiparticle entanglement of hot trapped ions". In: *Physical Review Letters* 82 (9 1999), pp. 1835–1838. DOI: 10.1103/PhysRevLett.82.1835.

[MSRH20] Giulia Meuli, Mathias Soeken, Martin Roetteler, and Thomas Häner. "Enabling accuracy-aware quantum compilers using symbolic resource estimation". In: *Proceedings of the ACM on Programming Languages* 4.OOPSLA (2020), 130:1–130:26. DOI: 10.1145/3428198.

[MSRM19] Giulia Meuli, Mathias Soeken, Martin Roetteler, and Giovanni De Micheli. "ROS: resource constrained oracle synthesis for quantum computers". In: [QPL20a], pp. 119–130. DOI: 10.4204/EPTCS.318.8.

[MSS07] Frédéric Magniez, Miklos Santha, and Mario Szegedy. "Quantum algorithms for the triangle problem". In: *SIAM Journal on Computing* 37.2 (2007), pp. 413–424. DOI: 10.1137/050643684.

[MSS13] Octavio Malherbe, Philip Scott, and Peter Selinger. "Presheaf models of quantum computation: an outline". In: *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky – Essays Dedicated to Samson Abramsky on the Occasion of His 60th Birthday*. Ed. by Bob Coecke, Luke Ong, and Prakash Panangaden. Vol. 7860. Lecture Notes in Computer Science. Springer, 2013, pp. 178–194. DOI: 10.1007/978-3-642-38164-5_13. arXiv: 1302.5652.

[MTT09] Paul-André Melliès, Nicolas Tabareau, and Christine Tasson. "An explicit formula for the free exponential modality of linear logic". In: *Proceedings of the 36th Internatilonal Colloquium on Automata, Languages and Programming, ICALP 2009, Part II* (Rhodes, Greece, July 5–12, 2009). Ed. by Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas. Vol. 5556. Lecture Notes in Computer Science. Springer, 2009, pp. 247–260. ISBN: 978-3-642-02929-5. DOI: 10.1007/978-3-642-02930-1_21. HAL: hal-00391714.

[MV06] M. Mottonen and J. J. Vartiainen. "Decompositions of general quantum gates". In: *Trends in Quantum Computing Research*. Ed. by Susan Shannon. Nova Science Publishers, 2006. Chap. 7. arXiv: quant-ph/0504100.

[MWD09] D. Michael Miller, Robert Wille, and Gerhard W. Dueck. "Synthesizing reversible circuits for irreversible functions". In: *Proceedings of the 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, DSD 2009* (Patras, Greece, Aug. 27–29, 2009). Ed. by Antonio Núñez and Pedro P. Carballo. IEEE Computer Society, 2009, pp. 749–756. ISBN: 978-0-7695-3782-5. DOI: 10.1109/DSD.2009.186.

[NC02] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002. ISBN: 0-521-63503-9.

[Nie97] M. A. Nielsen. "Computable functions, quantum measurements, and quantum dynamics". In: *Physical Review Letters* 79 (15 1997), pp. 2915–2918. DOI: 10.1103/PhysRevLett.79.2915. arXiv: quant-ph/9706006.

[Nik04] Rishiyur S. Nikhil. "BlueSpec System Verilog: efficient, correct RTL from high level specifications". In: *Proceedings of the 2nd ACM & IEEE International Conference on Formal Methods and Models for Co-Design, MEMOCODE 2004* (San Diego, California, USA, June 23–25, 2004). IEEE Computer Society, 2004, pp. 69–70. ISBN: 0-7803-8509-8. DOI: 10.1109/MEMCOD.2004.1459818.

[NM07] Paulin Jacobé de Naurois and Virgile Mogbil. "Correctness of multiplicative (and exponential) proof structures is *NL* -complete". In: *Proceedings of the 21st International Workshop on Computer Science Logic and 16th Annual Conference of the EACSL, CSL 2007* (Lausanne, Switzerland, Sept. 11–15, 2007). Ed. by Jacques Duparc and Thomas A. Henzinger. Vol. 4646. Lecture Notes in Computer Science. Springer, 2007, pp. 435–450. ISBN: 978-3-540-74914-1. DOI: 10.1007/978-3-540-74915-8_33. HAL: hal-00143926.

[Nor98] Michael Norrish. "C Formalised in HOL". Also: Technical Report UCAM-CL-TR-453. PhD thesis. University of Cambridge, 1998.

[NRSCM18] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. "Automated optimization of large quantum circuits with continuous parameters". In: *npj Quantum Information* 4.1 (2018), p. 23. DOI: 10.1038/s41534-018-0072-4.

[NST18] Rémi Nollet, Alexis Saurin, and Christine Tasson. "Local validity for circular proofs in linear logic with fixed points". In: *27th EACSL Annual Conference on Computer Science Logic, CSL 2018* (Birmingham, UK, Sept. 4–7, 2018). Ed. by Dan R. Ghica and Achim Jung. Vol. 119. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 35:1–35:23. ISBN: 978-3-95977-088-0. DOI: 10.4230/LIPIcs.CSL.2018.35. HAL: hal-01825477.

[NTR11] Michael Nachtigal, Himanshu Thapliyal, and Nagarajan Ranganathan. "Design of a reversible floating-point adder architecture". In: *Proceedings of the 11th IEEE International Conference on Nanotechnology*. 2011, pp. 451–456. DOI: 10.1109/NANO.2011.6144358.

[NV14] Trung Duc Nguyen and Rodney Van Meter. "A resource-efficient design for a reversible floating point adder in quantum computing". In: *ACM Journal on Emerging Technologies in Computing Systems* 11.2 (2014), 13:1–13:18. ISSN: 1550-4832. DOI: 10.1145/2629525. arXiv: 1306.3760.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering (ORFE). Springer, 2006.

[Öme00] Berhnard Ömer. "Quantum Programming in QCL". PhD thesis. TU Wien, 2000.

[Öme03] Berhnard Ömer. "Structured Quantum Programming". PhD thesis. TU Wien, 2003.

[Per08] Simon Perdrix. "Quantum entanglement analysis based on abstract interpretation". In: *Proceedings of the 15th International Symposium on Static Analysis (SAS'08)* (Valencia, Spain). Ed. by María Alpuente and Germán Vidal. Vol. 5079. Lecture Notes in Computer Science. Springer, 2008, pp. 270–282. DOI: 10.1007/978-3-540-69166-2_18. arXiv: 0801.4230.

[PHW06] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. "Reversible combinatory logic". In: *Mathematical Structures in Computer Science* 16.4 (2006), pp. 621–637. DOI: 10.1017/S0960129506005391.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002. ISBN: 0-262-16209-1.

[Pit13] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Vol. 57. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013. ISBN: 978-1-107-01778-8.

[PKI08] Sungwoo Park, Jinha Kim, and Hyeonseung Im. "Functional netlists". In: *Proceeding of the 13th ACM SIGPLAN international conference on Functional programming, ICFP 2008* (Victoria, BC, Canada, Sept. 20–28, 2008). Ed. by James Hook and Peter Thiemann. ACM, 2008, pp. 353–366. ISBN: 978-1-59593-919-7. DOI: 10.1145/1411204.1411253.

[Plo75] Gordon D. Plotkin. "Call-by-name, call-by-value and the lambda-calculus". In: *Theoretical Computer Science* 1.2 (1975), pp. 125–159. DOI: 10.1016/0304-3975(75)90017-1.

[Plo83] Gordon D. Plotkin. "Domains, Pisa Notes". Course notes on domain theory, available on the author's website under the name "Pisa Notes". 1983.

[PMAC+15] Lorenzo M. Procopio, Amir Moqanaki, Mateus Araújo, Fabio Costa, Irati Alonso Calafell, Emma G. Dowd, Deny R. Hamel, Lee A. Rozema, Časlav Brukner, and Philip Walther. "Experimental superposition of orders of quantum gates". In: *Nature Communications* 6.1 (2015), p. 7913. DOI: 10.1038/ncomms8913. arXiv: 1412.4006.

[PMMRT17] Christopher Portmann, Christian Matt, Ueli Maurer, Renato Renner, and Björn Tackmann. "Causal boxes: quantum information-processing systems closed under composition". In: *IEEE Transactions on Information Theory* 63.5 (2017), pp. 3277–3305. DOI: 10.1109/TIT.2017.2676805. arXiv: 1512.02240.

[PMSY+14] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. "A variational eigenvalue solver on a photonic quantum processor". In: *Nature* 5 (2014), p. 4213. DOI: 10.1038/ncomms5213.

[POPL14] Suresh Jagannathan and Peter Sewell, eds. *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'14* (San Diego, California, USA). ACM, 2014. ISBN: 978-1-4503-2544-8.

[POPL17] Giuseppe Castagna and Andrew D. Gordon, eds. *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL'17* (Paris, France). ACM, 2017. ISBN: 978-1-4503-4660-3. DOI: 10.1145/3009837.

[PPRZ20] Romain Péchoux, Simon Perdrix, Mathys Rennela, and Vladimir Zamdzhiev. "Quantum programming with inductive datatypes: causality and affine type theory". In: *Proceedings of the 23rd International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2020* (Dublin, Ireland, Apr. 25–30, 2020). Ed. by Jean Goubault-Larrecq and Barbara König. Vol. 12077. Lecture Notes in Computer Science. Springer, 2020, pp. 562–581. ISBN: 978-3-030-45230-8. DOI: 10.1007/978-3-030-45231-5_29. HAL: hal-03018513.

[PPZ19] Luca Paolini, Mauro Piccolo, and Margherita Zorzi. "qPCF: higher-order languages and quantum circuits". In: *Journal of Automated Reasoning* 63.4 (2019). Extended version of a TAMC'17 paper [PZ17]., pp. 941–966. DOI: 10.1007/s10817-019-09518-y.

[Pra81] Vaughan R. Pratt. "A decidable mu-calculus: preliminary report". In: *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (FOCS'81)* (Nashville, Tennessee, USA). IEEE Computer Society, 1981, pp. 421–427. DOI: 10.1109/SFCS.1981.4.

[Pre18] John Preskill. "Quantum computing in the NISQ era and beyond". In: *Quantum* 2 (2018), p. 79. DOI: 10.22331/q-2018-08-06-79. arXiv: 1801.00862v3.

[PRS17] Alex Parent, Martin Roetteler, and Krysta M. Svore. "REVS: a tool for space-optimized reversible circuit synthesis". In: [RC17], pp. 90–101. DOI: 10.1007/978-3-319-59936-6_7. arXiv: 1510.00377.

[PRZ17] Jennifer Paykin, Robert Rand, and Steve Zdancewic. "QWIRE: a core language for quantum circuits". In: [POPL17], pp. 846–858. DOI: 10.1145/3009837.3009894. HAL: hal-01474620. arXiv: 1610.09629.

[PS14] Adam Paetznick and Krysta M. Svore. "Repeat-until-success: non-deterministic decomposition of single-qubit unitaries". In: *Quantum Information and Computation* 14.15-016 (2014), pp. 1277–1301. DOI: 10.26421/QIC14.15-16-2. arXiv: 1311.1074.

[PSV14] Michele Pagani, Peter Selinger, and Benoît Valiron. "Applying quantitative semantics to higher-order quantum computing". In: [POPL14], pp. 647–658. DOI: 10.1145/2535838.2535879. arXiv: 1311.2290.

[pub22] ZX-calculus publications. *East, Richard and van de Wetering, John.* 2022. URL: https://zxcalculus.com/publications.html (visited on Aug. 24, 2022).

[PZ17] Luca Paolini and Margherita Zorzi. "qPCF: a language for quantum circuit computations". In: *Proceedings of the 14th Annual Conference on Theory and Applications of Models of Computation, TAMC'17* (Bern, Switzerland). Ed. by T. V. Gopal, Gerhard Jäger, and Silvia Steila. Vol. 10185. Lecture Notes in Computer Science. Extended journal version: [PPZ19]. 2017, pp. 455–469. ISBN: 978-3-319-55910-0. DOI: 10.1007/978-3-319-55911-7_33.

[QCS] *Official webpage for the IARPA research program QCS.* URL: https://www.iarpa.gov/research-programs/qcs (visited on Aug. 30, 2024).

[Qis] Qiskit Development Team. *Qiskit Documentation.* URL: https://qiskit.org/documentation/ (visited on July 28, 2021).

[QPL04] Peter Selinger, ed. *Proceedings of the Second International Workshop on Quantum Programming Languages, QPL'04* (Turku, Finland). Vol. 33. TUCS General Publication. TUCS, 2004. ISBN: 9-5212-1374-4. URL: http://urn.fi/URN:NBN:fi:bib:me:I00035039900.

[QPL07] Peter Selinger, ed. *Proceedings of the 3rd International Workshop on Quantum Programming Languages, QPL'05* (DePaul University, Chicago, USA). Vol. 170. Electronic Notes in Theoretical Computer Science. 2007.

[QPL08] Peter Selinger, ed. *Proceedings of the Fourth International Workshop on Quantum Programming Languages, QPL'06* (Oxford, UK.). Vol. 210. Electronic Notes in Theoretical Computer Science. July 2008.

[QPL11] B. Coecke, I. Mackie, P. Panangaden, and P. Selinger, eds. *Proceedings of the Joint 5th International Workshop on Quantum Physics and Logic and 4th Workshop on Developments in Computational Models, QPL/DCM 2008* (Reykjavik, Iceland). Vol. 270-1. Electronic Notes in Theoretical Computer Science. 2011.

[QPL14] Bob Coecke, Ichiro Hasuo, and Prakash Panangaden, eds. *Proceedings of the 11th workshop on Quantum Physics and Logic, QPL 2014* (Kyoto, Japan). Vol. 172. Electronic Proceedings in Theoretical Computer Science. 2014. DOI: 10.4204/EPTCS.172.

[QPL18] Bob Coecke and Aleks Kissinger, eds. *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017* (Nijmegen, The Netherlands). Vol. 266. Electronic Proceedings in Theoretical Computer Science. 2018.

[QPL19] Peter Selinger and Giulio Chiribella, eds. *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018* (Halifax, Canada, June 3–7, 2018). Vol. 287. EPTCS. 2019.

[QPL20a] Bob Coecke and Matthew Leifer, eds. *Proceedings 16th International Conference on Quantum Physics and Logic, QPL 2019* (Chapman University, Orange, CA, USA, June 10–14, 2019). Vol. 318. EPTCS. 2020.

[QPL20b] Benoît Valiron, Shane Mansfield, Pablo Arrighi, and Prakash Panangaden, eds. *Proceedings 17th International Conference on Quantum Physics and Logic, QPL 2020* (Online (due to Covid), June 2–6, 2020). Vol. 340. EPTCS. 2020.

[QPL23a] Stefano Gogioso and Matty Hoban, eds. *Proceedings 19th International Conference on Quantum Physics and Logic, QPL 2022* (Wolfson College, Oxford, UK, June 27–July 1, 2022). Vol. 394. EPTCS. 2023.

[QPL23b] Shane Mansfield, Benoît Valiron, and Vladimir Zamdzhiev, eds. *Proceedings of the Twentieth International Conference on Quantum Physics and Logic, QPL 2023* (Paris, France, July 17–21, 2023). Vol. 384. EPTCS. 2023. DOI: 10.4204/EPTCS.384.

[QTOOLS24] GQI. *List of Tools, by Quantum Computing Report*. 2024. URL: https://quantumcomputingreport.com/tools/ (visited on June 30, 2024).

[Qui20] Quingo Development Team. "Quingo: A Programming Framework for Heterogeneous Quantum-Classical Computing with NISQ Features". 2020. arXiv: 2009.01686.

[QZOO22] Stephen Jordan. *Quantum Algorithm Zoo*. 2022. URL: https://quantumalgorithmzoo.org/ (visited on Sept. 10, 2022).

[Ran14] André Ranchin. "Depicting qudit quantum mechanics and mutually unbiased qudit theories". In: [QPL14], pp. 68–91.

[Ran18] Robert Rand. "Formally Verified Quantum Programming". PhD thesis. University of Pennsylania, 2018. URL: https://repository.upenn.edu/edissertations/3175/ (visited on Aug. 29, 2021).

[RB01] Robert Raussendorf and Hans J. Briegel. "A one-way quantum computer". In: *Physical Review Letters* 86 (22 2001), pp. 5188–5191. DOI: 10.1103/PhysRevLett.86.5188. arXiv: quant-ph/0510135.

[RBB03] Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. "Measurement-based quantum computation on cluster states". In: *Physical Review A* 68 (2 2003), p. 022312. DOI: 10.1103/PhysRevA.68.022312. arXiv: quant-ph/0301052.

[RC13] Gerhard W. Dueck and D. Michael Miller, eds. *Proceedings of the 5th International Conference on Reversible Computation, RC'13* (Victoria, BC, Canada, July 4–5, 2013). Vol. 7948. Lecture Notes in Computer Science. Springer, 2013. ISBN: 978-3-642-38985-6. DOI: 10.1007/978-3-642-38986-3.

[RC17] Iain Phillips and Hafizur Rahaman, eds. *Proceedings of the 9th International Conference on Reversible Computation, RC'17* (Kolkata, India, July 6–7, 2017). Vol. 10301. Lecture Notes in Computer Science. Springer, 2017. ISBN: 978-3-319-59935-9. DOI: 10.1007/978-3-319-59936-6.

[RC18] Jarkko Kari and Irek Ulidowski, eds. *Proceedings of the 10th International Conference on Reversible Computation, RC 2018* (Leicester, UK). Vol. 11106. Lecture Notes in Computer Science. Springer, 2018. ISBN: 978-3-319-99497-0. DOI: 10.1007/978-3-319-99498-7.

[RC19] Michael Kirkedal Thomsen and Mathias Soeken, eds. *Proceedings of the 11th International Conference on Reversible Computation, RC 2019* (Lausanne, Switzerland). Vol. 11497. Lecture Notes in Computer Science. Springer, 2019. ISBN: 978-3-030-21499-9. DOI: 10.1007/978-3-030-21500-2.

[RC20] Ivan Lanese and Mariusz Rawski, eds. *Proceedings of the 12th International Conference on Reversible Computation, RC 2020*. Vol. 12227. Lecture Notes in Computer Science. Springer, 2020. ISBN: 978-3-030-52481-4. DOI: 10.1007/978-3-030-52482-1.

[RC21] Shigeru Yamashita and Tetsuo Yokoyama, eds. *Proceedings of the 13th International Conference on Reversible Computation, RC 2021* (Virtual Event, July 7–8, 2021). Vol. 12805. Lecture Notes in Computer Science. Springer, 2021. ISBN: 978-3-030-79836-9. DOI: 10.1007/978-3-030-79837-6.

[Reg04] Oded Regev. "Quantum computation and lattice problems". In: *SIAM Journal on Computing* 33.3 (2004), pp. 738–760. DOI: 10.1137/S0097539703440678.

[Reg92] Laurent Regnier. "Lambda-Calcul et Réseaux". Thèse de Doctorat. Université Paris 7, 1992.

[RG17] Lidia Ruiz-Perez and Juan Carlos García-Escartín. "Quantum arithmetic with the quantum fourier transform". In: *Quantum Information Processing* 16.6 (2017), p. 152. DOI: 10.1007/s11128-017-1603-1.

[RLNS00] K. Rustan, M. Leino, Greg Nelson, and James B. Saxe. *ESC/Java User's Manual*. SRC Technical Note 2000-002. Compaq Computer Corporation, 2000.

[Rob97] Ken Robinson. "The B method and the B toolkit". In: *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology, AMAST'97* (Sydney,Australia, Dec. 13–17, 1997). Ed. by Michael Johnson. Vol. 1349. Lecture Notes in Computer Science. Springer, 1997, pp. 576–580. ISBN: 3-540-63888-1. DOI: 10.1007/BFb0000503.

[Roe74] W. P. de Roever. "Recursive Program Schemes : Semantics and Proof Theory". Proefschrift. Vrije Universiteit te Amsterdam, 1974.

[Ros15] Neil J. Ross. "Algebraic and Logical Methods in Quantum Computation". PhD thesis. Dalhousie University, 2015. arXiv: 1510.02198.

[RP10] Jason Reed and Benjamin C. Pierce. "Distance makes the types grow stronger: a calculus for differential privacy". In: *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010* (Baltimore, Maryland, USA, Sept. 27–29, 2010). Ed. by Paul Hudak and Stephanie Weirich. ACM, 2010, pp. 157–168. ISBN: 978-1-60558-794-3. DOI: 10.1145/1863543.1863568.

[RPLZ18] Robert Rand, Jennifer Paykin, Dong-Ho Lee, and Steve Zdancewic. "Reqwire: reasoning about reversible quantum circuits". In: [QPL19], pp. 299–312. DOI: 10.4204/EPTCS.287.17.

[RS18a] Mathys Rennela and Sam Staton. "Classical control and quantum circuits in enriched category theory". In: *Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2018* (Dalhousie University, Halifax, Canada, June 6–9, 2018). Ed. by Sam Staton. Vol. 341. Electronic Notes in Theoretical Computer Science. See also the journal version [RS20]. Elsevier, 2018, pp. 257–279. DOI: 10.1016/j.entcs.2018.03.027. arXiv: 1711.05159.

[RS18b] Francisco Rios and Peter Selinger. "A categorical model for a quantum circuit description language". In: [QPL18], pp. 164–178. DOI: 10.4204/EPTCS.266.11. arXiv: 1706.02630.

[RS20] Mathys Rennela and Sam Staton. "Classical control, quantum circuits and linear logic in enriched category theory". In: *Logical Methods in Computer Science* 16.1 (2020). Journal version of an MFPS publication [RS18a]. DOI: 10.23638/LMCS-16(1:30)2020.

[RZBB94] Michael Reck, Anton Zeilinger, Herbert J. Bernstein, and Philip Bertani. "Experimental realization of any discrete unitary operator". In: *Physical Review Letters* 73 (1 July 1994), pp. 58–61. DOI: 10.1103/PhysRevLett.73.58.

[Saa03] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. 2nd ed. SIAM, 2003. ISBN: 978-0-89871-534-7.

[SB69] Dana Scott and J. W. de Bakker. "A Theory of Programs". Unpublished, manuscript notes for an IBM Seminar, Vienna, August 1969. Reprinted in [KMR89]. 1969.

[SBM06] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. "Synthesis of quantum-logic circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.6 (2006), pp. 1000–1010. DOI: 10.1109/TCAD.2005.855930. arXiv: quant-ph/0406176.

[Sch08] Maximilian A. Schlosshauer. *Decoherence and the Quantum-To-Classical Transition*. Springer, 2008.

[Sch14] Ulrich Schöpp. "Call-by-value in a basic logic for interaction". In: *Proceedings of the 12th Asian Symposium on Programming Languages and Systems,(APLAS 2014)* (Singapore). Ed. by Jacques Garrigue. Vol. 8858. Lecture Notes in Computer Science. Springer, Nov. 2014, pp. 428–448. ISBN: 978-3-319-12735-4. DOI: 10.1007/978-3-319-12736-1_23.

[Sch86] David A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn and Bacon, Inc, 1986.

[SCZ17] Robert S. Smith, Michael J. Curtis, and William J. Zeng. "A Practical Quantum Instruction Set Architecture". White paper presenting the assembly language Quil used in pyQuil for the Rigetti Forest framework. 2017. arXiv: 1608.03355v2.

[SDCSED20] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. "T|ket⟩: a retargetable compiler for NISQ devices". In: *Quantum Science and Technology* 6.1 (2020), p. 014003. DOI: 10.1088/2058-9565/ab8e92.

[Sel04a] Peter Selinger. "Towards a quantum programming language". In: *Mathematical Structures in Computer Science* 14.4 (2004), pp. 527–586. DOI: 10.1017/S0960129504004256.

[Sel04b] Peter Selinger. "Towards a semantics for higher-order quantum computation". In: [QPL04], pp. 127–143.

[Sel07] Peter Selinger. "Dagger compact closed categories and completely positive maps (extended abstract)". In: [QPL07], pp. 139–163. DOI: 10.1016/j.entcs.2006.12.018.

[SGLH11] Nikhil Swamy, Nataliya Guts, Daan Leijen, and Michael Hicks. "Lightweight monadic programming in ML". In: [ICFP11], pp. 15–27. DOI: 10.1145/2034773.2034778.

[SGTA+18] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. "Q#: enabling scalable quantum computing and development with a high-level DSL". In: *Proceedings of the Real World Domain Specific Languages Workshop 2018* (Vienna, Austria). ACM, 2018, 7:1–7:10. ISBN: 978-1-4503-6355-6. DOI: 10.1145/3183895.3183901. arXiv: 1803.00652.

[Sha70] D. F. Shanno. "Conditioning of quasi-Newton methods for function minimization". In: *Mathematics of Computation* 24 (1970), pp. 647–656.

[She94] Jonathan R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep. CMU-CS-94-125. School of Computer Science, Carnegie Mellon University, 1994.

[Sho94] Peter W. Shor. "Algorithms for quantum computation: discrete log and factoring". In: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS'94)* (Santa Fe, New Mexico, US.). IEEE. IEEE Computer Society Press, Nov. 1994, pp. 124–134. ISBN: 0-8186-6580-7. DOI: 10.1109/SFCS.1994.365700.

[Sho97] Peter W. Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172. arXiv: quant-ph/9508027.

[SHT18] Damian S. Steiger, Thomas Häner, and Matthias Troyer. "ProjectQ: an open source software framework for quantum computing". In: *Quantum* 2 (2018), p. 49. DOI: 10.22331/q-2018-01-31-49. arXiv: 1612.08091v2.

[Sim05] Alex K. Simpson. "Reduction in a linear lambda-calculus with applications to operational semantics". In: *Proceedings of the 16th International Conference on Term Rewriting and Applications, RTA'05* (Nara, Japan). Ed. by Jürgen Giesl. Vol. 3467. Lecture Notes in Computer Science. Springer, 2005, pp. 219–234. ISBN: 3-540-25596-6. DOI: 10.1007/978-3-540-32033-3_17.

[SJA17] Ingo Sander, Axel Jantsch, and Seyed-Hosein Attarzadeh-Niaki. "ForSyDe: system design using a functional language and models of computation". In: *Handbook of Hardware/Software Codesign*. Ed. by Soonhoi Ha and Jürgen Teich. Springer, 2017, pp. 99–140. ISBN: 978-94-017-7266-2. DOI: 10.1007/978-94-017-7267-9_5.

[SM13] Mehdi Saeedi and Igor L. Markov. "Synthesis and optimization of reversible circuits - a survey". In: *ACM Computing Surveys* 45.2 (2013), 21:1–21:34. DOI: 10.1145/2431211.2431220.

[SMB04] Vivek V. Shende, Igor L. Markov, and Stephen S. Bullock. "Minimal universal two-qubit controlled-NOT-based circuits". In: *Physical Review A* 69 (6 2004), p. 062321. DOI: 10.1103/PhysRevA.69.062321.

[SPA11] SPARK Team. *SPARK – The SPADE Ada Kernel (including RavenSPARK)*. 2011. URL: https://docs.adacore.com/sparkdocs-docs/SPARK_LRM.htm (visited on Sept. 10, 2022).

[SRSV14] Jonathan M. Smith, Neil J. Ross, Peter Selinger, and Benoît Valiron. "Quipper: concrete resource estimation in quantum algorithms". In: *Informal Proceedings of QAPL'14, Grenoble, France*. 2014. arXiv: 1412.0625.

[SRWD17] Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli. "Hierarchical reversible logic synthesis using luts". In: *Proceedings of the 54th Annual Design Automation Conference (DAC'17)* (Austin, TX, USA). ACM, 2017, 78:1–78:6. ISBN: 978-1-4503-4927-7. DOI: 10.1145/3061639.3062261.

[Sta15] Sam Staton. "Algebraic effects, linearity, and quantum programming languages". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'15* (Mumbai, India). Ed. by Sriram K. Rajamani and David Walker. ACM, 2015, pp. 395–406. ISBN: 978-1-4503-3300-9. DOI: 10.1145/2676726.2676999. URL: http://dl.acm.org/citation.cfm?id=2676726.

[Sto77] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey approach to Programming Language Theory*. Vol. 1. MIT Press Series in Computer Science. MIT Press, 1977.

[SV05] Peter Selinger and Benoît Valiron. "A lambda calculus for quantum computation with classical control". In: *Proceedings of the Seventh International Conference on Typed Lambda Calculi and Applications, TLCA'05* (Nara, Japan). Ed. by Pawel Urzyczyn. Vol. 3461. Lecture Notes in Computer Science. Journal version appeared in MSCS [SV06]. Springer Verlag, Apr. 2005, pp. 354–368. ISBN: 3-540-25593-1. DOI: 10.1007/11417170_26. HAL: hal-00483924. arXiv: cs/0404056.

[SV06] Peter Selinger and Benoît Valiron. "A lambda calculus for quantum computation with classical control". In: *Mathematical Structures in Computer Science* 16 (3 2006), pp. 527–552. DOI: 10.1017/S0960129506005238.

[SV08a] Peter Selinger and Benoît Valiron. "On a fully abstract model for a quantum linear functional language". In: [QPL08], pp. 123–137. DOI: 10.1016/j.entcs.2008.04.022.

[SV08b] Peter Selinger and Benoît Valiron. "A linear-non-linear model for a computational call-by-value lambda calculus (extended abstract)". In: *Proceedings of the 11th International Conference on Foundations of Software Science and Computational Structures, FoSSaCS'08* (Budapest, Hungary). Ed. by Roberto M. Amadio. Vol. 4962. Lecture Notes in Computer Science. Springer, 2008, pp. 81–96. DOI: 10.1007/978-3-540-78499-9_7. HAL: hal-00483903.

[SV09] Peter Selinger and Benoît Valiron. "Quantum lambda-calculus". In: [GM09]. Chap. 4, pp. 135–172.

[SVMABC17] Artur Scherer, Benoît Valiron, Siun-Chuon Mau, D. Scott Alexander, Eric van den Berg, and Thomas E. Chapuran. "Concrete resource analysis of the quantum linear-system algorithm used to compute the electromagnetic scattering cross section of a 2D target". In: *Quantum Information Processing* 16.3 (2017), p. 60. DOI: 10.1007/s11128-016-1495-5. HAL: hal-01474610. arXiv: 1505.06552.

[SVV18] Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. "From symmetric pattern-matching to quantum control". In: *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2018* (Thessaloniki, Greece). Ed. by Christel Baier and Ugo Dal Lago. Vol. 10803. Lecture Notes in Computer Science. Springer, 2018, pp. 348–364. DOI: 10.1007/978-3-319-89366-2_19. HAL: hal-01763568. arXiv: 1804.00952.

[SYG24] Razin A. Shaikh, Lia Yeh, and Stefano Gogioso. "The Focked-up ZX Calculus: Picturing Continuous-Variable Quantum Computation". Accepted for Communication at QPL 2024. 2024. arXiv: 2406.02905.

[TA15] Michael Kirkedal Thomsen and Holger Bock Axelsen. "Interpretation and programming of the reversible functional language RFUN". In: *Proceedings of the 27th Symposium on the Implementation and Application of Functional Programming Languages, IFL 2015, Koblenz, Germany, September 14-16, 2015*. Ed. by Ralf Lämmel. ACM, 2015, 8:1–8:13. ISBN: 978-1-4503-4273-5. DOI: 10.1145/2897336.2897345.

[Tai67] W. W. Tait. "Intensional interpretations of functionals of finite type". In: *Journal of Symbolic Logic* 32.2 (1967), pp. 198–212. DOI: 10.2307/2271658.

[TCMG+21] Márcio M. Taddei, Jaime Cariñe, Daniel Martínez, Tania García, Nayda Guerrero, Alastair A. Abbott, Mateus Araújo, Cyril Branciard, Esteban S. Gómez, Stephen P. Walborn, Leandro Aolita, and Gustavo Lima. "Computational advantage from the quantum superposition of multiple temporal orders of photonic gates". In: *PRX Quantum* 2 (1 2021), p. 010320. DOI: 10.1103/PRXQuantum.2.010320. arXiv: 2002.07817.

[TempHask] *Template Haskell Library*. URL: https://hackage.haskell.org/package/template-haskell (visited on Aug. 26, 2021).

[Tho12] Michael Kirkedal Thomsen. "A functional language for describing reversible logic". In: *Proceeding of the 2012 Forum on Specification and Design Languages (FDL'12)* (Vienna, Austria). IEEE, 2012, pp. 135–142. ISBN: 978-1-4673-1240-0. URL: http://ieeexplore.ieee.org/document/6336999/.

[TK05] Yasuhiro Takahashi and Noboru Kunihiro. "A linear-size quantum circuit for addition with no ancillary qubits". In: *Quantum Information and Computation* 5.6 (2005), pp. 440–448.

[TK08] Yasuhiro Takahashi and Noboru Kunihiro. "A fast quantum circuit for addition with few qubits". In: *Quantum Information and Computation* 8.6–7 (2008), pp. 636–649.

[TM22] Alex Townsend-Teague and Konstantinos Meichanetzidis. "Simplification Strategies for the Qutrit ZX-Calculus". Presentation at QPL 2022. 2022. arXiv: 2103.06914.

[Tof77] Tommaso Toffoli. "Computation and construction universality of reversible cellular automata". In: *Journal of Computer and System Sciences* 15.2 (1977), pp. 213–231. DOI: 10.1016/S0022-0000(77)80007-X.

[Tof80a] Tommaso Toffoli. *Reversible Computing*. Tech. rep. MIT/LCS/TM-151. See also the ICALP'80 paper [Tof80b]. MIT, 1980.

[Tof80b] Tommaso Toffoli. "Reversible computing". In: *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings* (Noordweijkerhout, The Netherlands, July 14–18, 1980). Ed. by J. W. de Bakker and Jan van Leeuwen. Vol. 85. Lecture Notes in Computer Science. See also the corresponding technical report [Tof80b]. Springer, 1980, pp. 632–644. ISBN: 3-540-10003-2. DOI: 10.1007/3-540-10003-2_104.

[Ton04] André van Tonder. "A lambda calculus for quantum computation". In: *SIAM Journal on Computing* 33.5 (2004), pp. 1109–1135. DOI: 10.1137/S0097539703432165. arXiv: quant-ph/0307150.

[Tra11] Paolo Tranquilli. "Intuitionistic differential nets and lambda-calculus". In: *Theoretical Computer Science* 412.20 (2011), pp. 1979–1997. DOI: 10.1016/j.tcs.2010.12.022.

[Tro92] Anne S. Troelstra. *Lectures in Linear Logic*. Vol. 29. CSLI Lecture Notes. Stanford, California, US.: Center for the Study of Language and Information, 1992. ISBN: 0-937073-77-6.

[Tur36] Alan M. Turing. "On computable numbers, with an application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society, Series 2* 42 (1936). Can be found integrally, and commented, in [Gir95b], pp. 230–265.

[Tur38] Alan Turing. "Systems of Logic Based on Ordinals". PhD thesis. Princeton University, 1938.

[Tur50] Alan M. Turing. "Computing machinery and intelligence". In: *Journal of the Mind Association* 59.236 (1950). Can be found integrally, and commented, in [Gir95b], pp. 433–460.

[Tur79] D. A. Turner. "A new implementation technique for applicative languages". In: *Software – Practice and Experience* 9.1 (1979), pp. 31–49. DOI: 10.1002/spe.4380090105.

[Unr19a] Dominique Unruh. "Quantum Hoare logic with ghost variables". In: [LICS19], pp. 1–13. DOI: 10.1109/LICS.2019.8785779.

[Unr19b] Dominique Unruh. "Quantum relational Hoare logic". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), 33:1–33:31. DOI: 10.1145/3290346.

[Val04] Benoît Valiron. "A Functional Programming Language for Quantum Computation With Classical Control". Master thesis. University of Ottawa, 2004. HAL: tel-00483944.

[Val08] Benoît Valiron. "Semantics for a Higher Order Functional Programming Language for Quantum Computation". PhD thesis. University of Ottawa, 2008. HAL: tel-00483944.

[Val10a] Benoît Valiron. "Orthogonality and algebraic lambda-calculus". In: *Proceedings of the 7th International QPL Workshop Quantum Physics and Logic, QPL'10* (Oxford, UK). Ed. by Bob Coecke, Prakash Panangaden, and Peter Selinger. 2010, pp. 169–175. URL: http://www.cs.ox.ac.uk/people/bob.coecke/QPL_proceedings.html.

[Val10b] Benoît Valiron. "Semantics of a typed algebraic lambda-calculus". In: *Proceedings of the Sixth Workshop on Developments in Computational Models: Causality, Computation, and Physics, DCM 2010* (Edinburgh, Scotland, July 9–10, 2010). Ed. by S. Barry Cooper, Prakash Panangaden, and Elham Kashefi. Vol. 26. Electronic Proceedings in Theoretical Computer Science. Preliminary work to the journal paper [Val13a]. 2010, pp. 147–158. DOI: 10.4204/EPTCS.26.14.

[Val11] Benoît Valiron. "On quantum and probabilistic linear lambda-calculi (extended abstract)". In: [QPL11], pp. 121–128. DOI: 10.1016/j.entcs.2011.01.011.

[Val12] Benoît Valiron. "Quantum computation: a tutorial". In: *New Generation Computing* 30.4 (2012), pp. 271–296. DOI: 10.1007/s00354-012-0401-7.

[Val13a] Benoît Valiron. "A typed, algebraic, computational lambda-calculus". In: *Mathematical Structures in Computer Science* 23.2 (2013). Journal, extended version of [Val10b]., pp. 504–554. DOI: 10.1017/S0960129512000205.

[Val13b] Benoît Valiron. "Quantum computation: from a programmer's perspective". In: *New Generation Computing* 31.1 (2013), pp. 1–26. DOI: 10.1007/s00354-012-0120-0.

[Val16] Benoît Valiron. "Generating reversible circuits from higher-order functional programs". In: *Proceedings of the 8th International Conference on Reversible Computation, RC'16* (Bologna, Italy). Ed. by Simon J. Devitt and Ivan Lanese. Vol. 9720. Lecture Notes in Computer Science. Springer, 2016, pp. 289–306. DOI: 10.1007/978-3-319-40578-0_21. HAL: hal-01474621.

[Val17] Benoît Valiron. *Programmer un ordinateur quantique.* Column in MathsInfos Hors-Série Numéro 3, published by Fondation Mathématique de Paris. 2017. HAL: hal-01763585.

[Val18] Benoît Valiron. "A formal analysis of quantum algorithms". In: *ERCIM News* 112 (Jan. 2018), pp. 23–24. HAL: hal-01763602.

[Val22] Benoît Valiron. "Semantics of quantum programming languages: classical control, quantum control". In: *Journal of Logical and Algebraic Methods in Programming* 128 (2022), p. 100790. DOI: 10.1016/J.JLAMP.2022.100790. HAL: hal-04038653.

[VAS06] Juliana K. Vizzotto, Thorsten Altenkirch, and Amr Sabry. "Structuring quantum effects: superoperators as arrows". In: *Mathematical Structures in Computer Science* 16.3 (2006), pp. 453–468. ARXIV: quant-ph/0501151.

[Vau09] Lionel Vaux. "The algebraic lambda-calculus". In: *Mathematical Structures in Computer Science* 19.5 (2009), pp. 1029–1059. DOI: 10.1017/S0960129509990089.

[VBE96] Vlatko Vedral, Adriano Barenco, and Artur Ekert. "Quantum networks for elementary arithmetic operations". In: *Physical Review A* 54.1 (1996), pp. 147–153. DOI: 10.1103/PhysRevA.54.147. ARXIV: quant-ph/9511018.

[Vil18] Renaud Vilmart. "A zx-calculus with triangles for Toffoli-Hadamard, Clifford+T, and beyond". In: [QPL19], 313–p344. DOI: 10.4204/EPTCS.287.18.

[Vil21] Renaud Vilmart. "The structure of sum-over-paths, its consequences, and completeness for Clifford". In: *Proceedings of the 24th International Conference on the Foundations of Software Science and Computation Structures, FoSSaCS 2021* (Luxembourg City, Luxembourg, Mar. 27–Apr. 1, 2021). Ed. by Stefan Kiefer and Christine Tasson. Vol. 12650. Lecture Notes in Computer Science. Springer, 2021, pp. 531–550. ISBN: 978-3-030-71994-4. DOI: 10.1007/978-3-030-71995-1_27. HAL: hal-02651473.

[VKB21] Augustin Vanrietvelde, Hlér Kristjánsson, and Jonathan Barrett. "Routed quantum circuits". In: *Quantum* 5 (2021), p. 503. DOI: 10.22331/q-2021-07-13-503. ARXIV: 2011.08120.

[VRSAS15] Benoît Valiron, Neil J. Ross, Peter Selinger, Dana Scott Alexander, and Jonathan M. Smith. "Programming the quantum future". In: *Communications of the ACM* 58.8 (2015), pp. 52–61. DOI: 10.1145/2699415. URL: http://doi.acm.org/10.1145/2699415. HAL: hal-01194416.

[VZ14a] Benoît Valiron and Steve Zdancewic. "Finite vector spaces as model of simply-typed lambda-calculi". In: *Proceedings of the 11th International Colloquium on Theoretical Aspects of Computing, ICTAC 2014* (Bucharest, Romania, Sept. 17–19, 2014). Ed. by Gabriel Ciobanu and Dominique Méry. Vol. 8687. Lecture Notes in Computer Science. See [VZ14b] for the long version. Springer, 2014, pp. 442–459. DOI: 10.1007/978-3-319-10882-7_26.

[VZ14b] Benoît Valiron and Steve Zdancewic. "Modeling simply-typed lambda calculi in the category of finite vector spaces". In: *Scientific Annals of Computer Science* 24.2 (2014), pp. 325–368. DOI: 10.7561/SACS.2014.2.325.

[Wad03] Philip Wadler. "Call-by-value is dual to call-by-name". In: *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP'03* (Uppsala, Sweden). Ed. by Colin Runciman and Olin Shivers. ACM, 2003, pp. 189–201. ISBN: 1-58113-756-7. DOI: 10.1145/944705.944723.

[Wad93] Philip Wadler. "A syntax for linear logic". In: [MFPS93], pp. 513–529.

[Wan17] Quanlong Wang. "Qutrit ZX-calculus is complete for stabilizer quantum mechanics". In: [QPL18], pp. 58–70. DOI: 10.4204/EPTCS.266.3.

[WB14] Quanlong Wang and Xiaoning Bian. "Qutrit dichromatic calculus and its universality". In: [QPL14], pp. 92–101. DOI: 10.4204/EPTCS.172.7.

[WB89] Philip Wadler and Stephen Blott. "How to make ad-hoc polymorphism less ad-hoc". In: *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages (POPL'89)* (Austin, Texas, USA, Jan. 11–13, 1989). ACM Press, 1989, pp. 60–76. DOI: 10.1145/75277.75283.

[WBA11] James D. Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. "Simulation of electronic structure Hamiltonians using quantum computers". In: *Molecular Physics* 109.5 (2011), pp. 735–750. DOI: 10.1080/00268976.2011.552441.

[WC20] Matt Wilson and Giulio Chiribella. "A Diagrammatic Approach to Information Transmission in Generalised Switches". 2020. arXiv: 2003.08224.

[WD10] Robert Wille and Rolf Drechsler. "Effect of BDD optimization on synthesis of reversible and quantum logic". In: *Proceedings of the Workshop on Reversible Computation, RC'09* (York, UK). Ed. by Irek Ulidowski. Vol. 253. Electronic Notes in Theoretical Computer Science 6. 2010, pp. 57–70. DOI: 10.1016/j.entcs.2010.02.006.

[WDAB21] Julian Wechs, Hippolyte Dourdent, Alastair A. Abbott, and Cyril Branciard. "Quantum circuits with classical versus quantum control of causal order". In: *PRX Quantum* 2 (3 2021), p. 030335. DOI: 10.1103/PRXQuantum.2.030335. HAL: hal-03124176. arXiv: 2101.08796.

[Wes16] Abraham Westerbaan. "Quantum programs as kleisli maps". In: *Proceedings of the 13th International Conference on Quantum Physics and Logic, QPL 2016* (Glasgow, Scotland). Ed. by Ross Duncan and Chris Heunen. Vol. 236. Electronic Proceedings in Theoretical Computer Science. 2016, pp. 215–228. DOI: 10.4204/EPTCS.236.14.

[Wes19] Abraham Anton Westerbaan. "The Category of Von Neumann Algebras". PhD thesis. Radboud Universiteit Nijmegen, 2019. URL: https://hdl.handle.net/2066/201611. arXiv: 1804.02203.

[WGTDD08] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W. Dueck, and Rolf Drechsler. "RevLib: an online resource for reversible functions and reversible circuits". In: *Proceedings of the 38th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2008* (Dallas, Texas, USA, May 22–23, 2008). IEEE Computer Society, 2008, pp. 220–225. ISBN: 978-0-7695-3155-7. DOI: 10.1109/ISMVL.2008.43.

[Win80] Glynn Winskel. "Events in Computation". PhD thesis. University of Edinburgh, 1980.

[Win87] Glynn Winskel. "Event structures". In: *Petri Nets: Applications and Relationships to Other Models of Concurrency*. Ed. by W. Brauer, W. Reisig, and G. Rozenberg. Springer Berlin Heidelberg, 1987, pp. 325–392. ISBN: 978-3-540-47926-0.

[WK13] Nathan Wiebe and Vadym Kliuchnikov. "Floating point representations in quantum circuit synthesis". In: *New Journal of Physics* 15.9 (2013), p. 093041. DOI: 10.1088/1367-2630/15/9/093041. arXiv: 1305.5528.

[WOD10] Robert Wille, Sebastian Offermann, and Rolf Drechsler. "SyReC: a programming language for synthesis of reversible circuits". In: *Proceedings of the 2010 Forum on specification & Design Languages, FDL 2010, September 14-16, 2010, Southampton, UK*. Ed. by Adam Morawiec and Jinnie Hinderscheit. See also Journal's version [WSSD16]. ECSI, Electronic Chips & Systems design Initiative, 2010, pp. 184–189. DOI: 10.1049/ic.2010.0150.

[WR16] Nathan Wiebe and Martin Roetteler. "Quantum arithmetic and numerical analysis using repeat-until-success circuits". In: *Quantum Information and Computation* 16.1&2 (2016), pp. 134–178. DOI: 10.26421/QIC16.1-2-9. arXiv: 1406.2040.

[WS14] Dave Wecker and Krysta M. Svore. "LIQUi|⟩: A Software Design Architecture and Domain-Specific Language for Quantum Computing". 2014. arXiv: 1402.4467.

[WSSD16] Robert Wille, Eleonora Schönborn, Mathias Soeken, and Rolf Drechsler. "SyReC: a hardware description language for the specification and synthesis of reversible circuits". In: *Integration, the VLSI Journal* 53 (2016). See also extended abstract presented at FDL'10 [WOD10]., pp. 39–53. DOI: 10.1016/j.vlsi.2015.10.001.

[Wüt11] Adrian Wüthrich. *The Genesis of Feynman Diagrams*. Vol. 26. Archimedes. Springer, 2011.

[WY22] John van de Wetering and Lia Yeh. "Building qutrit diagonal gates from phase gadgets". In: [QPL23a]. DOI: 10.4204/EPTCS.394.4.

[XVY21] Zhaowei Xu, Benoît Valiron, and Mingsheng Ying. "Reasoning about Recursive Quantum Programs". Draft, to appear in ACM TOCL. 2021. arXiv: 2107.11679.

[YAG12] Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. "Towards a reversible functional language". In: *Revised Papers of the Third International Workshop on Reversible Computation, RC'11* (Gent, Belgium, July 4–5, 2011). Ed. by Alexis De Vos and Robert Wille. Vol. 7165. Lecture Notes in Computer Science. Springer, 2012, pp. 14–29. DOI: 10.1007/978-3-642-29517-1_2.

[YAG16] Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. "Fundamentals of reversible flowchart languages". In: *Theoretical Computer Science* 611 (2016), pp. 87–115. DOI: 10.1016/j.tcs.2015.07.046.

[Yao93] A. Chi-Chih Yao. "Quantum circuit complexity". In: *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS'93)* (Washington, DC, USA). IEEE Computer Society, 1993, pp. 352–361. DOI: 10.1109/SFCS.1993.366852.

[YG07] Tetsuo Yokoyama and Robert Glück. "A reversible programming language and its invertible self-interpreter". In: *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, PEPM 2007, Nice, France, January 15-16, 2007*. Ed. by G. Ramalingam and Eelco Visser. 2007, pp. 144–153. DOI: 10.1145/1244381.1244404.

[Yin11] Mingsheng Ying. "Floyd-Hoare logic for quantum programs". In: *ACM Transactions on Programming Languages and Systems* 33.6 (2011), 19:1–19:49. DOI: 10.1145/2049706.2049708. arXiv: 0906.4586.

[Yin19] Mingsheng Ying. "Toward automatic verification of quantum programs". In: *Formal Aspects of Computing* 31.1 (2019), pp. 3–25. DOI: 10.1007/s00165-018-0465-3. arXiv: 1807.11610.

[YLYF14] Mingsheng Ying, Yangjia Li, Nengkun Yu, and Yuan Feng. "Model-checking linear-time properties of quantum systems". In: *ACM Transactions on Computational Logic* 15.3 (2014), 22:1–22:31. DOI: 10.1145/2629680.

[YYFD13] Mingsheng Ying, Nengkun Yu, Yuan Feng, and Runyao Duan. "Verification of quantum programs". In: *Science of Computer Programming* 78.9 (2013), pp. 1679–1700. DOI: 10.1016/j.scico.2013.03.016.

[YYW17] Mingsheng Ying, Shenggang Ying, and Xiaodi Wu. "Invariants of quantum programs: characterisations and generation". In: [POPL17], pp. 818–832. DOI: 10.1145/3009837.3009840.

[YZLF22] Mingsheng Ying, Li Zhou, Yangjia Li, and Yuan Feng. "A proof system for disjoint parallel quantum programs". In: *Theoretical Computer Science* 897 (2022), pp. 164–184. DOI: 10.1016/j.tcs.2021.10.025.

[ZF10] Dongsheng Zhao and Taihe Fan. "Dcpo-completion of posets". In: *Theoretical Computer Science* 411.22-24 (2010), pp. 2167–2173. DOI: 10.1016/j.tcs.2010.02.020.

[ZW17] Alwin Zulehner and Robert Wille. "Improving synthesis of reversible circuits: exploiting redundancies in paths and nodes of QMDDs". In: [RC17], pp. 232–247. DOI: 10.1007/978-3-319-59936-6_18.

[ZYC20] Xiaobin Zhao, Yuxiang Yang, and Giulio Chiribella. "Quantum metrology with indefinite causal order". In: *Physical Review Letters* 124 (19 2020), p. 190503. DOI: 10.1103/PhysRevLett.124.190503.

[ZYY19] Li Zhou, Nengkun Yu, and Mingsheng Ying. "An applied quantum Hoare logic". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019* (Phoenix, AZ, USA, June 22–26, 2019). Ed. by Kathryn S. McKinley and Kathleen Fisher. ACM, 2019, pp. 1149–1162. ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314584.

Index