The Geometry of Quantum Compilation

Anonymous author
Anonymous affiliation

Abstract

We introduce a compilation algorithm turning any term of a linear quantum λ-calculus into a quantum circuit with classical wires. The essential ingredient of the proposed algorithm is Girard's geometry of interaction, which, differently from its well-known uses from the literature, is here leveraged to anticipate the *classical* computation as much as possible, while producing a circuit that, when executed, corresponds to the underlying *quantum* part of the λ-term. Noticeably, the circuit construction takes time linear in the size of the underlying type derivation, without incurring the exponential blowup encountered when using plain operational semantics.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Lambda Calculus, Quantum Computation, Geometry of Interaction, Quantum Compilation

1 Introduction

Quantum computing holds immense potential to revolutionize various areas of computer science by solving complex problems much faster than classical computing [26, 42]. This is due to its ability to leverage the power of quantum bits, thanks to the principles of *superposition* and *entanglement*. One of the fascinating aspects of quantum computing is the diversity of model architectures being explored. These include gate-based quantum computing, but also quantum annealing [29], topological quantum computing [30], and others.

On the side of high-level quantum programming languages, the QRAM model of quantum computation [31] is certainly one of the most successful ones. There, a classical computing machine interacts with a quantum processor by instructing the latter to create new qubits, apply some unitary transformations to the existing qubits, or measure (some of) them. In other words, computation consists of a sequence of interactions between the classical machine and the quantum processor, similarly to what happens when programming in presence of an external storage device. Thanks to measurements, however, the overall evolution (and the computation's final result) can be probabilistic. Moreover, the classical computer and the quantum processor do not follow the same rules: while the former is a purely classical device, the latter's internal state consists of a finite number of qubits, each of them being manipulated following the laws of quantum mechanics. In particular, quantum bits cannot be erased nor duplicated, and the operations the quantum processor can perform are of a very specific shape.

Based on this model, several *quantum programming languages* have been developed, from assembly code [10, 9], to imperative programming languages with loops and classical tests [15], to functional programming languages and λ-calculi [41]. All these languages share the same core principle: they manipulate an external quantum memory and can apply quantum operations to it. As the program is executed, the quantum memory is updated until it reaches a final state. As such, then, they precisely follow the QRAM model. In particular, between any pair of interaction points, the underlying classical machine can perform an arbitrary amount of work. More recently, a different family of programming languages [35, 14, 3, 6, 14] is instead more focused on what happens *inside* the quantum memory and allow the user to write custom-made quantum operations through so-called quantum conditionals.

However, there is a fracture between the aforementioned programming paradigms, and the reality of quantum computing. Most quantum architectures are currently very hard to be accessed interactively, requiring *a whole circuit* to be sent to them. The latter is first created in its entirety by a classical algorithm, and then processed and optimized for the specific architecture. As a result, quantum programs are, in practice, often written down either as programs representing *one* quantum circuit [10], or as programs in so-called *circuit description languages*, like `Qiskit` [28], `Cirq` [43], or `Quipper` [24, 23]. These languages manipulate circuits as ordinary data structures, often taking the form of libraries for mainstream programming languages. This separates circuit construction from circuit execution, enabling optimization and transpilation. Programming is thus seen as the direct construction of quantum circuits and the model is no longer the one of QRAM: the program's task consists in building a circuit, and not of interacting with a quantum device.

It is thus natural to ask whether it is possible to reconcile the approach where programming follows the QRAM model with the need to produce complete circuits, thereby being able to target existing hardware architectures. This work can be seen as giving an answer to the following question: would it be possible to compile QRAM languages, and in particular functional languages with higher-order functions [41], down to quantum circuits, this way anticipating the classical work and postponing as much as possible the quantum operations? Moreover, would it be possible to do that *efficiently* in presence of both measurements and conditionals? As detailed in Section 2 below, this is not trivial and cannot be easily solved by relying on the usual, rewriting based operational semantics of the underlying language. A conditional whose branches have a higher-order type, indeed, cannot be easily and efficiently compiled to a conditional of the target circuit language: as soon as a branching is *opened*, it is difficult to unify the two branches, effectively *closing* the branching. This can give rise to an exponential blowup in the size of the underlying circuit, a well-known phenomenon which shows up, e.g., in the related problem of dynamically lifting [16] the value of a Boolean in circuit description languages.

Contributions. In this work we propose a new compilation procedure for the linear fragment of Selinger & Valiron quantum λ-calculus [41] onto a paradigmatic quantum circuit model corresponding to a fragment of the QASM language [10]. Notably, our procedure gets rid of *both* higher-order operations and classical control, and thus provides an effective approach to answer the following question: given some well-typed term of first-order type in which higher-order functions and conditionals can possibly occur, is it possible to efficiently compute the underlying circuit, effectively anticipating as much as possible the classical work? The fundamental ingredient of our compilation scheme is Girard's *Geometry of Interaction* (GoI for short) [20, 19], a semantic framework for linear logic proofs which is has been variously used to derive suitable *abstract machines* [36, 13, 1] and circuit *synthesis* algorithms [17]. The basic idea of the GoI translation can be described as follows: given a type derivation π with conclusion $\Gamma \vdash M : A$, one introduces a finite set of tokens which can travel along occurrences of base types in Γ and A throughout the type derivation; the paths followed by such tokens then give rise to a circuit relating *positive* and *negative* occurrences of ground types in the conclusion of π. In our approach, we consider typing derivations for a linear quantum λ-calculus, and, by following the paths of tokens a quantum circuit is progressively produced. In these circuits, the inputs and outputs corresponds respectively to the negative and positive occurrences of the types `bit` and `qbit`. For instance, a type derivation of $x : \texttt{qbit}, y : \texttt{qbit} \vdash M : \texttt{qbit} \otimes \texttt{qbit}$ (where M might indeed contain higher-order operations as well as conditionals) will, after compilation, give rise to a standard quantum circuit with two input qubits and two output qubits.

Our compilation procedure works in two steps: first, the typing derivation is translated via GoI onto a language for quantum circuits *with* classical control. Then, a second compilation step takes place, which translates the circuit onto a QASM circuit, notably *eliminating* the use of classical control flow. The whole compilation scheme works in time linear in the size of the underlying type derivation π. As described in Section 7, the approach is robust enough to be adapted to calculi with graded monads. After presenting the compilation procedure, we establish its soundness: we prove that, whenever a term M of the linear quantum λ-calculus, on a given input state $|\phi\rangle$, produces a distribution of states after the (probabilistic) rewriting, then the circuit produced by compiling M, on input state $|\phi\rangle$, will produce the same distribution. To this purpose, we exploit a simulation result with respect to another GoI interpretation of quantum λ-calculi [11], but also relying on the *completely positive maps* interpretation of quantum circuits [39].

An extended version of this paper is available as supplementary material.

2 A Bird's Eye View on the Problem

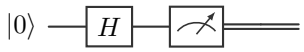
This section aims at informally introducing the challenges addressed in this paper, while at the same time analyzing the difficulties that lie ahead.

Suppose we are dealing with the following term, written in a standard quantum λ-calculus akin to those introduced in the literature:

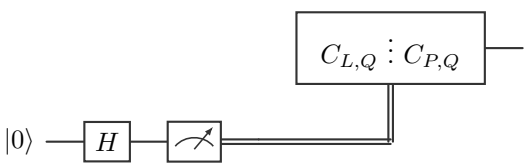
$$M := \texttt{let}\, x = (\texttt{if}\ N\ \texttt{then}\ L\ \texttt{else}\ P)\ \texttt{in}\ Q.$$

Suppose further that the branches L and P have higher-order type and that the guard N is a term of Boolean type whose value is produced through some form of quantum computation. Consider, for example, the case where N is $\texttt{meas}(H(\texttt{new ff}))$, while L and P are $\lambda x.x$ and $\lambda x.H(x)$, respectively. Here, H stands for the *Hadamard* gate of quantum computing, while `new` and `meas` initializes and measure a qubit, respectively.

Suppose we want to compile M into an equivalent quantum circuit. We could proceed by executing M using the operational semantics of the underlying language, in the form of a reduction relation or an abstract machine. In evaluating N, such semantics would not perform any quantum operations, but would produce a circuit in the following form:



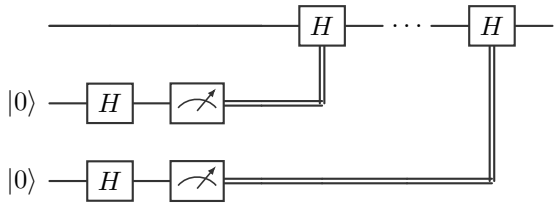
As the Boolean produced by N depends on a measurement, its value would not be known at circuit-*building* time, but only at circuit-*execution* time. Therefore, it is clear that both branches of the conditional instruction $\texttt{if}\ N\ \texttt{then}\ L\ \texttt{else}\ P$ should be executed. The two branches, however, are two values having functional types, so how could we proceed? If we wanted to keep the machine we are defining compliant with the reduction semantics, it would be natural to proceed by evaluating $Q[x \leftarrow L]$ and $Q[x \leftarrow P]$ onto two circuits $C_{Q,L}$ and $C_{Q,P}$. The overall circuit constructed would then have the following form:



Why is this way of proceeding problematic? The point is that there is a risk of an exponential blowup in the size of the produced circuit. Suppose we generalize the above example to a family of terms $M_n = R_n^n$, where

$$R_{n+1}^m := \texttt{let}\, x_n = (\texttt{if}\ N\ \texttt{then}\ L\ \texttt{else}\ P)\ \texttt{in}\ R_n^m; \qquad R_0^m = \lambda x.x_1(x_2(\ldots(x_m x))).$$

It is clear that by proceeding as above while compiling M_n, we would obtain a circuit of exponential size in n consisting of n levels of nested conditionals. It is equally clear, however, that there is a simpler circuit which corresponds to the term M_n, namely the following:



The technique we propose in this work goes precisely in this direction and allows us to compile the terms M_n into the circuit above, thus avoiding the exponential blowup arising from standard operational semantics.

3 Calculus

In this section we briefly describe the syntax and operational semantics of the λ-calculus which we consider as a source language, which we call λ^Q. This is a linear version of Selinger and Valiron's quantum λ-calculus, [41], almost identical to the one considered in [32].

The language consists of the standard linear λ-calculus, with pairs and `let` constructions, as well as Booleans and conditionals. Quantum operators are available as term constants. We define the set of terms, noted Λ^Q, as follows:

$$\begin{aligned}
\Lambda^Q \ni M, N, P ::= {} & x \mid \lambda x.M \mid M\ N && \text{Function Operators}\\
& \mid \langle M, N\rangle \mid \texttt{let}\,\langle x, y\rangle = M\ \texttt{in}\ N && \text{Pair Operators}\\
& \mid \texttt{tt} \mid \texttt{ff} \mid \texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P && \text{Booleans \& Conditionals}\\
& \mid c && \text{Quantum Operations}
\end{aligned}$$

Here, x ranges over an infinite set of variables, while c ranges over a finite set $\mathbb{Q}$ of term constants. We sometimes use standard syntactic sugar, e.g. $\texttt{let}\, x = M\ \texttt{in}\ N$ or $\lambda x_1, \ldots, x_n.M$. *Values* can be defined in the natural way. The set $\mathbb{Q}$ includes three kinds of quantum operations: `new`, that creates a qubit from a bit, `meas`, that implements the so-called measurement operation, and operators implementing a universal set of quantum gates (e.g., the set $\{H, S, T, \text{CNOT}\}$, also called Clifford + T).

▶ **Example 1.** A term QCF implementing a fair quantum coin-flip can be written down as $QCF := \texttt{meas}(\text{H}(\texttt{new ff}))$. This is precisely the term N we considered in the introduction.

The λ^Q-calculus comes equipped with a type system based on linear logic [18]. In this type system, every piece of data has to be used exactly once. This means that nothing can be duplicated nor erased, hence respecting the laws of quantum physics regarding the non-duplication of arbitrary data. The type system contains two kind of base types, the classical bits (denoted `bit`) and quantum bits (denoted `qbit`). They can then be combined through tensors or function types:

$$A, B ::= \texttt{qbit} \mid \texttt{bit} \mid A \multimap B \mid A \otimes B$$

$$\frac{}{x : A \vdash x : A} \qquad \frac{\Delta \vdash M : A \multimap B \qquad \Gamma \vdash N : A}{\Delta, \Gamma \vdash M\ N : B} \qquad \frac{\Delta \vdash M : \texttt{bit} \qquad \Gamma \vdash N : A \qquad \Gamma \vdash P : A}{\Delta, \Gamma \vdash \texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P : A}$$

Figure 1 Examples of typing rules of λ^Q.

A type context (denoted Γ, Δ) is a set of pairs of variables and their corresponding types. We denote by $\mathbb{B}$ the base types: $\mathbb{B} ::= \texttt{bit} \mid \texttt{qbit}$. The map $\mathcal{T}$ attributes types to the operators in $\mathbb{Q}$ in the natural way, e.g. $\mathcal{T}(\text{CNOT}) = \texttt{qbit} \otimes \texttt{qbit} \multimap \texttt{qbit} \otimes \texttt{qbit}$, $\mathcal{T}(\texttt{meas}) = \texttt{qbit} \multimap \texttt{bit}$, and $\mathcal{T}(\texttt{new}) = \texttt{bit} \multimap \texttt{qbit}$.

Typing is itself standard. Judgments take the form $\Gamma \vdash M : A$, where Γ is a typing context. An excerpt of the typing rules is in Figure 1 (see [5] for the details):

▶ **Example 2.** The term QCF from Example 1 is a well-typed term of type $\texttt{bit}$ under the empty context.

The calculus λ^Q can be naturally endowed with a *call-by-value* reduction semantics. While linear functions and pairs can be treated in a completely standard way, quantum data and quantum operations require some care. In order to manipulate them, λ^Q makes use of an external quantum register in which the qubits the underlying program manipulates are stored. Each variable representing a quantum data is linked to a certain qubit in the quantum memory. When applying a quantum operation to these e variables, the qubits corresponding to those variables will be modified accordingly. This is formalized through the notion of a quantum closure.

A *quantum closure*, (or simply *closure*). This is a triple $[Q, L, M]$ where $L = [x_1, \ldots, x_n]$ is a list of variables, and Q is a normalized vector of $\mathbb{C}^{2^n}$. The *evaluation contexts*, indicated with metavariables like E, are defined in the natural way, and as usual, $E[M]$ is the term we obtain from E by filling the hole $[\cdot]$ with the term M. We write $M[x \leftarrow N]$ for the capture-avoiding substitution of x in M by the term N. The operational semantics of λ^Q, following [41], is probabilistic and generated by relations $M \to_p N$, for $p \in [0, 1]$, corresponding to the fact that M reduces to N with probability p. Such relations are generated by three kinds of rules. First of all, we have standard deterministic rules for the λ-calculus, which are defined for closures but which only act on their third components, e.g.,

$$[Q, L, (\lambda x.M)V] \to_1 [Q, L, M[x \leftarrow V]]$$
$$[Q, L, \texttt{if tt then}\ M\ \texttt{else}\ N] \to_1 [Q, L, M]$$

Then, there are rules which serve to give meaning to the operators in QCF, e.g.,

$$[Q, L, U\langle x_{j_1}, \ldots, x_{j_n}\rangle] \to_1 [R, L, \langle x_{j_1}, \ldots, x_{j_n}\rangle]$$
$$[Q, L, \texttt{new tt}] \to_1 [Q \otimes |1\rangle, L \cup \{y\}, y]$$
$$[\alpha |Q_0\rangle + \beta |Q_1\rangle, L, \texttt{meas}\ x] \to_{|\alpha|^2} [|Q_0\rangle, L, \texttt{ff}]$$
$$[\alpha |Q_0\rangle + \beta |Q_1\rangle, L, \texttt{meas}\ x] \to_{|\beta|^2} [|Q_1\rangle, L, \texttt{tt}]$$

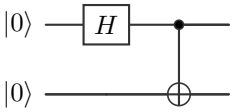
where R is obtained from Q by applying the gate U to the qubits $j_1, \ldots, j_n$, while $|Q_0\rangle$ and $|Q_1\rangle$ are normalized states of the form $\sum_j \alpha_j |\psi_j^i\rangle \otimes |i\rangle \otimes |\phi_j^i\rangle$ for $i \in \{0, 1\}$, respectively. Finally, we have a rule for congruence:

$$[Q, L, E[M]] \to_p [R, J, E[N]] \text{ whenever } [Q, L, M] \to_p [R, J, N]$$

We denote by $M \to_p^* N$ the existence of terms $M_1 = M, M_2, \ldots, M_n = N$ and reals $p_1, \ldots, p_{n-1} \in [0, 1]$ such that $M_i \to_{p_i} M_{i+1}$, for $i = 1, \ldots, n-1$ and $p = \prod_i p_i$.

$$
\begin{aligned}
[-,-,M] &= [-,-,(\lambda x.\,\mathrm{CNOT}(H\ x, \texttt{new ff}))\ (\texttt{new ff})] \\
&\to_1 [|0\rangle, [y], (\lambda x.\,\mathrm{CNOT}(H\ x, \texttt{new ff}))\ y] && \text{Evaluate } \texttt{new ff} \\
&\to_1 [|0\rangle, [y], \mathrm{CNOT}(H\ y, \texttt{new ff})] && \text{Substitute } y \\
&\to_1 [\frac{|0\rangle + |1\rangle}{\sqrt{2}}, [y], \mathrm{CNOT}(y, \texttt{new ff})] && \text{Evaluate } H\ y \\
&\to_1 [\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle, [y, z], \mathrm{CNOT}(y, z)] && \text{Evaluate } \texttt{new ff} \\
&\to_1 [\frac{|00\rangle + |11\rangle}{\sqrt{2}}, [y, z], (y, z)] && \text{Evaluate CNOT}
\end{aligned}
$$

(a) Example of evaluation of a closure.



(b) Corresponding circuit.

Figure 2 Evaluation in λ^Q.

We say that a quantum closure $[Q, L, M]$ is of type A under context Γ, denoted $\Gamma \vdash [Q, L, M] : A$ if $L = [x_1, \ldots, x_n]$ and $\Gamma, x_1 : \texttt{qbit}, \ldots, x_n : \texttt{qbit} \vdash M : A$ is a valid typing judgement.

We can prove basic results like a substitution lemma, type preservation and progress in a standard way (see, again [5] for some more details). As an example, subject reduction can be formulated as follows: if $\Gamma \vdash [Q, L, M] : A$ and $[Q, L, M] \to_p [R, J, N]$ then $\Gamma \vdash [R, J, N] : A$. Progress instead becomes the following: given a well-typed quantum closure $\vdash [Q, L, M] : A$, either M is a value or $[Q, L, M] \to_p [Q', L', M']$. Finally, we can also prove that reduction is normalizing. As a consequence, any closure $[Q, L, M]$ uniquely determines a finite distribution $\mathrm{eval}_\lambda[Q, L, M] = \sum_{i=1}^{k} p_i Q_i$ of quantum states, so that $[Q, L, M] \to^*_{p_i} [Q_i, L_i, V]$.

▶ **Example 3.** Consider the term $M = (\lambda f.\lambda x.\,\mathrm{CNOT}(f\ x, \texttt{new ff}))\ (\texttt{new ff}) : \texttt{qbit} \otimes \texttt{qbit}$. We illustrate its reduction in Figure 2a, where Q and L are initially empty (as there are no free variables in M). This term computes the Bell's State (also call EPR pair), notice that we used higher-order functions to change the state of the second qubit by feeding another argument instead of `new ff`. The quantum circuit representing this example is the one in Figure 2b, and in Section 5, we will show how the compiling procedure produces it.

4 Quantum Circuits

In this section we introduce a language for quantum circuits with classical if-then-else, called $\mathcal{QC}$, and we describe its interpretation in the compact closed category of *completely positive maps* [39, 40]. Then, we show that any circuit is equivalent to one without if-then-else, by describing a procedure to eliminate the use of classical control flow.

4.1 Quantum Circuits with Classical Control Flow

Let $\mathbb{L}$ be an infinite set of *labels*, indicated as l, r, to be used as names for the wires of circuits. The types and terms (called *circuits*) of $\mathcal{QC}$ are defined below:

$$
\begin{aligned}
\mathbb{B} &::= \texttt{bit} \mid \texttt{qbit} && \text{Base Types} \\
A, B &::= \mathbb{B} \mid A \otimes B && \text{Types} \\
C, D &::= U^{\Gamma}_{\Delta} \mid C; D \mid \texttt{if } C \texttt{ then } D \texttt{ else } E && \text{Circuits}
\end{aligned}
$$

where $U \in \mathbb{Q}$ and the environment are defined as $\Gamma, \Delta ::= \emptyset \mid \Gamma \cup \{l : \mathbb{B}\}$. We require that any label occurs at most once in an environment. We denote as $\Gamma \otimes \Delta$ the union of environments Γ and Δ, with the proviso that Γ, Δ have no label in common. The expression

$$\frac{}{\emptyset \rhd 0 \rhd \texttt{bit}}\ 0 \qquad \frac{}{\emptyset \rhd 1 \rhd \texttt{bit}}\ 1 \qquad \frac{U_\Delta^\Gamma \in \mathbb{Q}}{\Theta_1 \otimes \Gamma \otimes \Theta_2 \rhd U_\Delta^\Gamma \rhd \Theta_1 \otimes \Delta \otimes \Theta_2}\ \texttt{Q}$$

$$\frac{\Gamma \rhd C \rhd \Psi \qquad \Psi \rhd D \rhd \Delta}{\Gamma \rhd C; D \rhd \Delta}\ ; \qquad \frac{\Gamma_1 \rhd C \rhd \texttt{bit} \otimes \Phi \qquad \Gamma_2 \rhd D \rhd \Delta \qquad \Gamma_2 \rhd E \rhd \Delta}{\Gamma_1 \otimes \Gamma_2 \rhd \texttt{if}\ C\ \texttt{then}\ D\ \texttt{else}\ E \rhd \Phi \otimes \Delta}\ \texttt{ite}$$

Figure 3 Circuit typing rules.

U_Δ^Γ indicates a gate from Γ to Δ, which we may also note as $U : \Gamma \to \Delta$. For example, considering l_1, l_2, l_3, l_4 labels pointing to some qubits, we can have $\mathrm{CNOT}^{\{l_1:\texttt{qbit}, l_2:\texttt{qbit}\}}_{\{l_3:\texttt{qbit}, l_4:\texttt{qbit}\}}$.

A *type judgement* for a circuit C is an expression of the form $\Gamma \rhd C \rhd \Delta$. Type judgements are deduced via the rules in Fig. 3. Notice that, in the rule Q, if U is from Γ to Δ, then it can be typed with additional contexts Θ_1, Θ_2. Intuitively, the corresponding circuit is the tensorized gate $\mathrm{Id}_{\Theta_1} \otimes U \otimes \mathrm{Id}_{\Theta_2}$; in other words, we will consider each gate as acting on *all qubits*, and not just on a subset of the available qubits. This allows us to not consider an operator for parallel composition.

We interpret circuits in terms of density matrices and completely positive maps, as in [40]. The category **CPM** has for objects finite tuples of positive numbers $\sigma = (n_1, \dots, n_k)$ and as morphisms completely positive maps $V_\sigma \to V_{\sigma'}$ where $V_{(n_1,\dots,n_k)} = \mathbb{C}^{n_1 \times n_1} \times \cdots \times \mathbb{C}^{n_k \times n_k}$. **CPM** is symmetric monoidal closed (in fact, dagger compact closed, cf. [41]), with tensor product $(n_1, \dots, n_k) \otimes (m_1, \dots, m_k) = (n_1 m_1, \dots, n_k m_k)$ and an isomorphism $V_{\sigma\otimes\tau} \equiv V_\sigma \otimes V_\tau$ as well as a unit $\mathbb{1} = (1)$. Via the natural isomorphism $\Phi_{A,B,C} : \mathbf{CPM}(\rho \otimes \sigma, \tau) \to \mathbf{CPM}(\rho, \sigma \otimes \tau)$, sending $f \in \mathbf{CPM}(\rho \otimes \sigma, \tau)$ into $g(s) = \sum_{b \in B(V_\sigma)} b \otimes f(s \otimes b)$, where $B(V_\sigma)$ is a basis for V_σ, the tensor product is also the hom-object of **CPM**.

We interpret each type of $\mathcal{QC}$ as an object of **CPM** via $[\![\texttt{bit}]\!] = (1,1)$, $[\![\texttt{qbit}]\!] = (2)$ and $[\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!]$. The interpretation of an environment Γ is given by $[\![\emptyset]\!] = \mathbb{1}$ and $[\![\Gamma \cup \{l : \mathbb{B}\}]\!] = [\![\Gamma]\!] \otimes [\![\mathbb{B}]\!]$.

In order to interpret the typing judgments we need an interpretation of the gates $U \in \mathbb{Q}$. For each gate U_Δ^Γ, we define the function $[\![U]\!] : (1) \to [\![\Gamma \multimap \Delta]\!]$ as follows:

- for each quantum gate $U : \texttt{qbit}^n \to \texttt{qbit}^n$, with $U \in \{H, S, T, \mathrm{CNOT}\}$, we define $[\![U : \texttt{qbit}^n \multimap \texttt{qbit}^n]\!] = \Phi(x \mapsto \widehat{U} x \widehat{U^\dagger}) : (1) \to [\![\texttt{qbit}]\!]^n \otimes [\![\texttt{qbit}]\!]^n$, where we use U also to indicate the corresponding linear map $U : \mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$, and where $\widehat{U} \in \mathbb{C}^{2^n \times 2^n}$ indicates the matrix given by $\widehat{U}_{ij} = e_i \cdot (U e_j)$.
- $[\![0]\!](x) = (x, 0) : (1) \to [\![\texttt{bit}]\!]$ and $[\![1]\!](x) = (0, x) : (1) \to [\![\texttt{bit}]\!]$,
- $[\![\mathrm{new}]\!] = \Phi(\iota) : (1) \to [\![\texttt{bit}]\!] \otimes [\![\texttt{qbit}]\!]$, where $\iota : (1,1) \to (2)$ is the map $(a, b) \to \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$,
- $[\![\mathrm{meas}]\!] = \Phi(p) : (1) \to [\![\texttt{qbit}]\!] \otimes [\![\texttt{bit}]\!]$, where $p : (2) \to (1,1)$ is the map $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \to (a, d)$,

Finally, for any type judgment $\Gamma \rhd C \rhd \Delta$ we define a completely positive linear map $[\![C]\!] : [\![\Gamma]\!] \to [\![\Delta]\!]$ by induction, as illustrated in Fig. 4, where in the last rule we used the fact that $[\![\texttt{bit}]\!] \otimes [\![\Phi]\!]$ is isomorphic to $[\![\Phi]\!] \times [\![\Phi]\!]$, so that $f : [\![\Gamma_1]\!] \to [\![\texttt{bit}]\!] \otimes [\![\Phi]\!]$ can be split into two maps $f_0, f_1 : [\![\Gamma_1]\!] \to [\![\Phi]\!]$.

4.2 Eliminating Classical Control Flow

We now show how, for any $\mathcal{QC}$-circuit C, it is possible to eliminate all uses of the if-then-else constructor, thus producing a standard quantum circuit that computes the same completely positive map.

$$[\![\emptyset \rhd 0 \rhd \mathtt{bit}]\!](x) = (x, 0) \qquad [\![\emptyset \rhd 1 \rhd \mathtt{bit}]\!](x) = (0, x) \qquad \frac{\begin{array}{c}[\![\Gamma \rhd C \rhd \Psi]\!] = f : [\![\Gamma]\!] \to [\![\Psi]\!] \\ [\![\Psi \rhd D \rhd \Delta]\!] = g : [\![\Psi]\!] \to [\![\Delta]\!]\end{array}}{[\![\Gamma \rhd C; D \rhd \Delta]\!] = g \circ f : [\![\Gamma]\!] \to [\![\Delta]\!]}$$

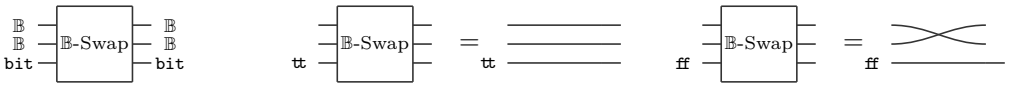
$$\frac{U_\Delta^\Gamma \in \mathbb{Q}}{[\![\Theta_1 \otimes \Gamma \otimes \Theta_2 \rhd U_\Delta^\Gamma \rhd \Theta_1 \otimes \Delta \otimes \Theta_2]\!] = \mathrm{Id}^{|\Theta_1|} \otimes [\![U]\!] \otimes \mathrm{Id}^{|\Theta_2|}}$$

$$\frac{\begin{array}{c}[\![\Gamma_1 \rhd C \rhd \mathtt{bit} \otimes \Phi]\!] = f : [\![\Gamma_1]\!] \to [\![\mathtt{bit}]\!] \otimes [\![\Phi]\!] \\ [\![\Gamma_2 \rhd D \rhd \Delta]\!] = g_1 : [\![\Gamma_2]\!] \to [\![\Delta]\!] \\ [\![\Gamma_2 \rhd E \rhd \Delta]\!] = g_2 : [\![\Gamma_2]\!] \to [\![\Delta]\!]\end{array}}{\begin{array}{r}[\![\Gamma_1, \Gamma_2 \rhd \mathtt{if}\ C\ \mathtt{then}\ D\ \mathtt{else}\ E \rhd \Phi \otimes \Delta]\!](x \otimes y) = (f_0(x) \otimes g_1(y)) + (f_1(x) \otimes g_2(y)) \\ : [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \to [\![\Phi]\!] \otimes [\![\Delta]\!]\end{array}}$$

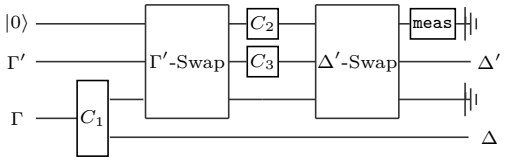
Figure 4 From type judgement to completely positive maps.

▶ **Theorem 4.** *For any circuit $\Gamma \rhd C \rhd \Delta$ of size n there exists an if-then-else-free circuit $\Gamma \rhd D \rhd \Delta$ of size $\mathcal{O}(n)$ such that $[\![C]\!] = [\![D]\!]$.*

Proof sketch. For each base type $\mathbb{B} = \mathtt{bit}, \mathtt{qbit}$ one can easily construct an if-then-else circuit $\mathbb{B}$-Swap as below left, satisfying the two equations below right



By sequentially composing the circuits $\mathbb{B}$-Swap one can then, for any environment Γ, construct a circuit $\mathtt{bit} \otimes \Gamma \rhd \Gamma\text{-Swap} \rhd \mathtt{bit} \otimes \Gamma$ that behaves in a similar way. Then, we can replace any sub-circuit of C of the form $\Gamma \otimes \Gamma' \rhd \mathtt{if}\ C_1\ \mathtt{then}\ C_2\ \mathtt{else}\ C_3 \rhd \Delta \otimes \Delta'$, where $\Gamma \rhd C_1 \rhd \mathtt{bit} \otimes \Delta$ and $\Gamma' \rhd C_2, C_3 \rhd \Delta'$, with the circuit below



where the gate ⫴ indicates a discarded bit. Observe that, if C_1 produces the bit $\mathtt{tt}$, then the circuit above will compute C_2 and "kill" C_3, that is, feed it with $|0\rangle$ and measure and discard its result; conversely, if C_1 produces the bit $\mathtt{ff}$, then the circuit above will similarly compute C_3 and "kill" C_2. ◀

5 The Quantum Circuit Token Machine

In this section we introduce the quantum circuit machine QCSIAM (Quantum Circuit Synchronous Interaction Abstract Machine), a machine that translates any quantum λ-term into a circuit in $\mathcal{QC}$. By post-processing this circuit as shown at the end of Section 4, we can thus obtain a standard quantum circuit. We define the machine in two steps: we first introduce a machine QCSIAM_0 for the if-then-else free fragment of λ^Q; then we introduce the full machine, whose execution requires to make *several* runs of the QCSIAM_0.

5.1 The If-Then-Else-Free Machine

The QCSIAM_0 is a *token machine*, in the style of Girard's geometry of interaction [20, 19, 13, 4]. The idea is that, starting from a type derivation π in λ^Q, we will let a finite set of *tokens*

move through π, following the occurrences of *base types* `bit` or `qbit`. Each token starts from a *negative* occurrence of some base type in π, corresponding to an input, and eventually reaches a *positive* occurrence of some base type, corresponding to an output. As they travel along the derivation, the tokens capture the underlying circuit of the λ^Q-term. At each step of the execution of the QCSIAM_0, each token lies in a position inside π, corresponding to an occurrence of some base type $\mathbb{B}$ in some type judgement occurring in π. To be more precise, let us first introduce the notion of *position within a type* A. Any occurrence of some base type $\mathbb{B}$ in a type A is determined by a unique *occurrence context* $\mathtt{C}$, corresponding intuitively to a type containing a unique occurrence of the hole $[-]$, so that $A = \mathtt{C}[\mathbb{B}]$. Occurrence contexts are defined by:

$$\mathtt{C} ::= [-] \mid \mathtt{C} \multimap A \mid A \multimap \mathtt{C} \mid \mathtt{C} \otimes A \mid A \otimes \mathtt{C}$$

The position is called *positive* or *negative* depending on whether $\mathtt{C}$ belongs to the positive or negative contexts, defined as follows:

$$\mathcal{P} ::= [-] \mid \mathcal{N} \multimap A \mid A \multimap \mathcal{P} \mid \mathcal{P} \otimes A \mid A \otimes \mathcal{P}$$
$$\mathcal{N} ::= \mathcal{P} \multimap A \mid A \multimap \mathcal{N} \mid \mathcal{N} \otimes A \mid A \otimes \mathcal{N}$$

A *position within a judgement* $\Gamma \vdash A$ is either a position in A or a position in some type B occurring in Γ. If the position is within A, then it is positive (resp. negative) precisely when it is positive (resp. negative) as a position within A; if the position is within B, then it is positive (resp. negative) when it is negative (resp. positive) as a position within B. Finally, a *position within a type derivation* π is a position within some judgement occurring in π, and its polarity corresponds to its polarity as a position within the judgement.

We can now define tokens:

▶ **Definition 5** (Token). *A* token *in a type derivation π is a pair (l, σ) where $l \in L$ is a label and σ is a position in π.*

Given a derivation $\pi : \Gamma \vdash M : A$, three sets of tokens (which are unique up to relabeling) will play an important role:

- the set of tokens in the negative positions of the conclusion $\Gamma \vdash A$, noted Π_π^-;
- the set of tokens in the positive positions of the conclusion $\Gamma \vdash A$, noted Π_π^+;
- the set of tokens in the positions of axioms $\vdash \mathtt{tt}, \mathtt{ff} : \mathtt{bit}$, noted Π_π^{bit}.

Intuitively, the tokens Π_π^- will correspond to the *inputs* of the circuit, while the tokens Π_π^+ will correspond to the *outputs*. The tokens Π_π^{bit} will also be important for the initialization of the machine. We will indicate as Δ_π^- (resp. Δ_π^+, $\Delta_\pi^{\mathtt{bit}}$) the circuit environments formed by labels and base types of the tokens in Π_π^- (resp. Π_π^+, $\Pi_\pi^{\mathtt{bit}}$).

The machine QCSIAM_0 is actually a *multitoken* machine. This means that many tokens move inside π. During the execution, their paths will produce a circuit. A *configuration* of QCSIAM_0 is given by the positions of the tokens together with the circuit produced so far. More precisely, a configuration $\mathcal{C}$ is a tuple $(\pi, \mathcal{M}, C)$ where:

- $\pi : \Gamma \vdash M : A$ is a type derivation of λ^Q;
- $\mathcal{M}$ is a set of tokens;
- $\Delta_\pi^- \rhd C \rhd \Delta_{\mathcal{M}}$ is a circuit of $\mathcal{QC}$ with inputs the negative positions in the conclusion of π and outputs $\Delta_{\mathcal{M}}$ corresponding to the base types in the positions of the tokens $\mathcal{M}$.

In a run of the QCSIAM_0 the tokens move from the positions Π_π^- and eventually reach the positive positions Π_π^+, hence producing a circuit $\Delta_\pi^- \rhd C \rhd \Delta_\pi^+$. More precisely, a

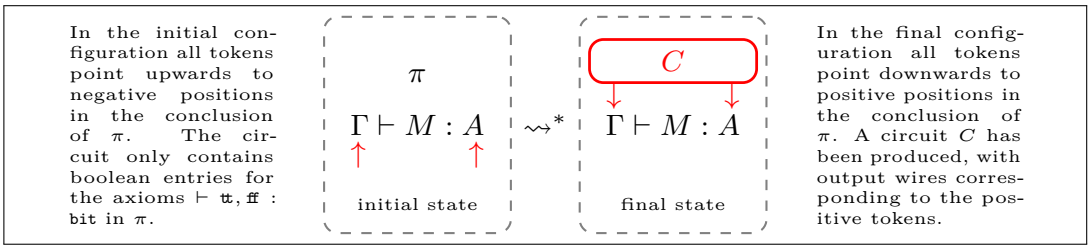
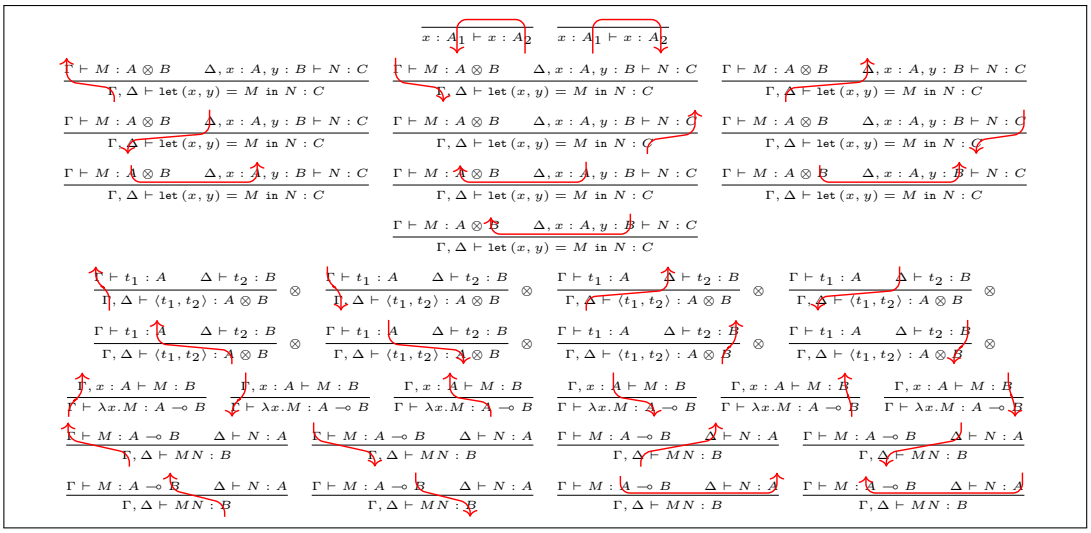
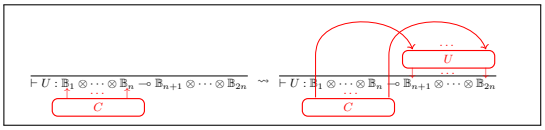


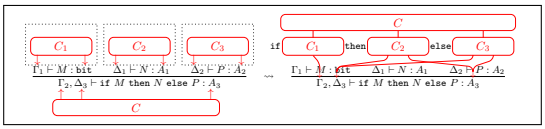
Figure 5 Execution of the quantum circuit token machine.



(a) Structural rules.



(b) Circuit rule.



(c) If-then-else rule.

Figure 6 Informal description of the rules of QCSIAM.

configuration $\mathcal{C} = (\pi, \mathcal{M}, \Delta_\pi^- \rhd C \rhd \Delta_\mathcal{M})$ is *initial* when $\mathcal{M} = \Pi_\pi^- \cup \Pi_\pi^{\mathtt{bit}}$, $\Delta_\mathcal{M} = \Delta_\pi^- \cup \Delta_\pi^{\mathtt{bit}}$ and the circuit C is formed by all bit-gates corresponding to the axioms $\vdash \mathtt{tt}, \mathtt{ff} : \mathtt{bit}$ in π tensored with identities on all other wires. Instead, a configuration $\mathcal{C} = (\pi, \mathcal{M}, C)$ is *final* when $\mathcal{M} = \Pi_\pi^+$. This is illustrated in Figure 5, where the arrows pointing upwards describe token in negative positions, which move upwards in the derivation, while the arrows pointing downwards describe token in positive positions, which move downwards.

▶ Remark 6. The situation we are interested in is that of a derivation $\pi : \Gamma \vdash M : A$ where the types in Γ and A are *first-order*, that is, do not contain the linear arrow $\multimap$. In this case the initial positions coincide with the base types in Γ, and the final positions coincide with the base types in A. We call this a *circuit typing*. Observe that a circuit typing is possibly the type of a $\mathcal{QC}$-term. The token machine will then produce a circuit $\bigotimes \Gamma \rhd C \rhd A$. Also observe that in a circuit typing the types that do not occur in the conclusion of the derivation may still contain linear arrows.

The token dynamics are described by two kinds of rules. In the *structural rules*, see

Figure 6a, one of the tokens moves upwards or downwards across the type derivation, while the circuit is left unchanged. These rules are as in standard GoI interpretations of the linear λ-calculus. Notice that there is no rule for `tt` and `ff`, this is because their rules (the initialization of a token) is already taken care of when we initialize the token machine for the first time. Instead, in the *circuit rule*, illustrated in Figure 6b, the tokens move in a *synchronized* way: after *all* necessary tokens reach the negative occurrences of $\mathbb{B}_1, \dots, \mathbb{B}_n$, *all* such tokens move onto the positive occurrences $\mathbb{B}_{n+1}, \dots, \mathbb{B}_{2n}$; when this happens, the gate U on the corresponding wires is composed with the circuit C constructed so far, while keeping the other wires unchanged.

▶ **Example 7.** Consider the term $M = x : \texttt{qbit}, y : \texttt{qbit} \vdash \mathrm{CNOT}(Hx, y) : \texttt{qbit} \otimes \texttt{qbit}$. We illustrate the run of the token machine over its type derivation in Fig. 7. If we replace the two variables x, y by `new ff` one can see that the produced circuit coincides with the one from Example 3 (cf. Figure 2).

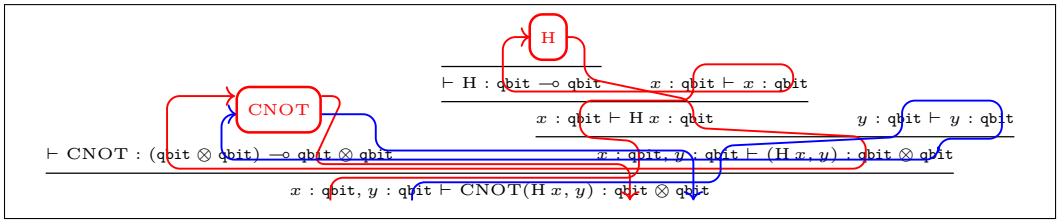


Figure 7 Example of execution of the token machine without if-then-else.

The result below shows that the QCSIAM_0 reaches a final configuration in linear time.

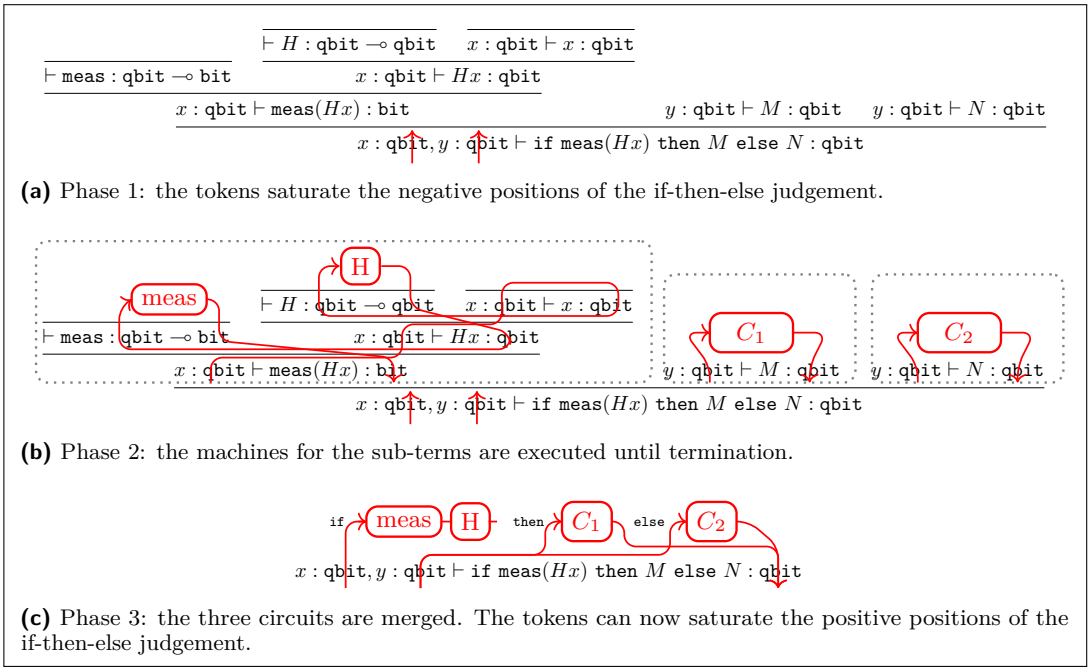
▶ **Proposition 8** (QCSIAM_0). *For any derivation $\pi : \Gamma \vdash M : A$ where M is if-then-else-free, any run of the QCSIAM_0 reaches a final configuration in $O(|\pi|)$ steps, producing a circuit $\Delta_\pi^- \rhd C \rhd \Delta_\pi^+$.*

Proof. This can be proved by a standard GoI argument, cf. [21]. The fundamental observation is that the paths of any single token is always *acyclic*, that is, it never visits the same position twice. Since all paths are thus necessarily finite, and may only terminate in a final position (by inspection of the rules), all tokens eventually reach a final position after any available position has been visited exactly once. ◀

5.2 The Full Machine

We now introduce the full machine, also accounting for the if-then-else operator. The idea is to let *multiple* QCSIAM_0 interact in a hierarchical way: suppose that, during the run of a machine m, the tokens saturate, moving upwards, the negative positions in some judgement $J = \Gamma \vdash \texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P : A$; three new machines are then launched, corresponding to the type derivations of the subterms M, N, P; once the tokens of the three machines have reached their final positions, producing three circuits C_1, C_2, C_3, the machine m merges these circuits to produce the controlled circuit $\texttt{if}\ C_1\ \texttt{then}\ C_2\ \texttt{else}\ C_3$, and moves its tokens towards the positive positions in J. Observe that, with this architecture, the machine m remains stuck until the other three machines have reached a final position. We will actually show that m may only get stuck for a finite amount of time, and that the whole hierarchical execution always terminates in linear time.

The idea sketched above is at work in the rule for the if-then-else, illustrated in Fig. 6c and discussed in the example below.



(a) Phase 1: the tokens saturate the negative positions of the if-then-else judgement.

(b) Phase 2: the machines for the sub-terms are executed until termination.

(c) Phase 3: the three circuits are merged. The tokens can now saturate the positive positions of the if-then-else judgement.

Figure 8 Three phases of the execution of an if-then-else rule.

▶ **Example 9.** Let $x : \texttt{qbit} \vdash M, N : \texttt{qbit}$ be two terms and consider the term $T = \texttt{if}\ P\ \texttt{then}\ M\ \texttt{else}\ N$, where $P = \texttt{meas}(\texttt{H}x)$, corresponding to a fair flip coin. The execution of the token machine over T works in three phases, illustrated in Figure 8.

The following result shows that any run of the QCSIAM terminates in linear time on a final configuration:

▶ **Theorem 10.** *For any derivation $\pi : \Gamma \vdash M : A$, any run of the QCSIAM reaches a final configuration in $O(|\pi|)$ steps, producing a circuit $\Delta_\pi^- \rhd C \rhd \Delta_\pi^+$.*

Proof. We argue by induction on the maximum number $\kappa(M)$ of nested occurrences of if-then-else in M. If $\kappa(M) = 0$, then M is if-then-else-free, so we conclude by Proposition 8. Otherwise, let us first show that the machine never gets stuck: if the tokens reach an if-then-else $\texttt{if}\ N\ \texttt{then}\ P\ \texttt{else}\ Q$, the corresponding three sub-derivations must be such that $\kappa(N), \kappa(P), \kappa(Q) < \kappa(N)$; by induction hypothesis, then, the three machines terminate in a final configuration, so that the if-then-else rule can be applied to move the tokens away from $\texttt{if}\ N\ \texttt{then}\ P\ \texttt{else}\ Q$. Now one can argue for the termination of the machine in a similar way to the proof of Proposition 8, since all paths are necessarily finite. ◀

6 Soundness of the Quantum Circuit Token Machine

In this section we prove that the QCSIAM is sound with respect to the underlying operational semantics of λ^Q , that is, that the circuits produced by the machine perfectly match the quantum protocols described by the corresponding λ^Q-terms.

The situation we are interested in is the one in which we are given a circuit typing $\Gamma \vdash M : A$ for some λ^Q-term M. Let us first highlight an important difference between λ^Q and the QCSIAM. On the one hand, the evaluation of λ^Q-terms relies on the choice of a quantum register (a closure) and is intrinsically probabilistic, due to the measure

operator; in other words, the different evaluations of a closure $[Q, L, M]$ produce a *distribution* $\mathrm{eval}_\lambda[Q, L, M]$ of quantum states. On the other hand, the execution of the token machine does not rely on a quantum register and is entirely deterministic: the measure operator results in the introduction in the circuit of a measure gate that is *not* evaluated at compile time. In particular, the evaluation of the QCSIAM over M does not produce a distribution of quantum states, but a quantum circuit C_M.

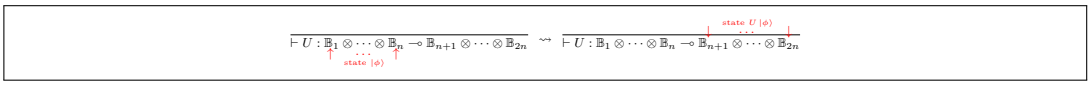
The soundness of the token machine must thus be formulated as a result relating the association $Q \mapsto \mathrm{eval}_\lambda[Q, L, M]$ between quantum states and distributions of quantum states (i.e. *mixed* states) produced by the evaluation of the quantum closure, with the action of the quantum circuit C_M (or rather, of its **CPM**-interpretation) over mixed states. For any finite distribution of quantum states $\mu = \sum_{i=1}^{k} p_i Q_i$ over n qubits, and $L = [x_1, \ldots, x_n]$, let $\mathrm{state}(\mu, L) \in [\![\mathtt{qbit}^n]\!]$ indicate the associated mixed state. This leads to the following:

▶ **Theorem 11.** *Let $\pi : \Gamma \vdash M : A$ be a circuit typing derivation and suppose that the QCSIAM produces, over π, the final configuration $(\pi, \Pi_\pi^-, \Delta_\pi^- \rhd C_M \rhd \Delta_\pi^+)$. Then, for any quantum closure $[Q, L, M]$, $[\![C_M]\!](\mathrm{state}(Q, L)) = \mathrm{state}(\mathrm{eval}_\lambda[Q, L, M])$.*

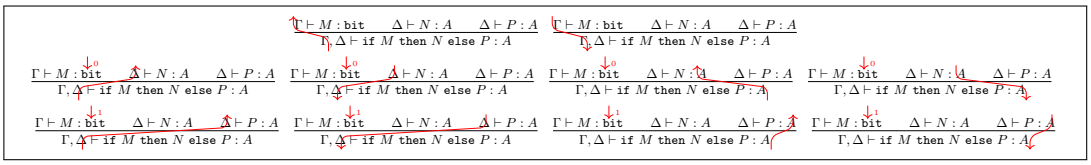
We provide a sketch of the proof of Theorem 11. The fundamental ingredient is the introduction of yet another token machine, called the QMSIAM (Quantum Memory-based Synchronous Interaction Abstract Machine) whose execution is closer to the evaluation of λ^Q-terms, being probabilistic. This machine is essentially the same as the machine MSIAM of [32, 11]. Also in the QMSIAM we have multiple tokens that move from initial positions to final positions inside the type derivation of M; however, the machine does *not* produce a circuit; instead, the machine has access to a quantum register $[Q, L]$ that is updated during execution. This quantum register contains the current states of the qubits and bits in the positions occupied by the tokens (for uniformity, we can suppose that the states $b \in \{0, 1\}$ of the bit are registered as the corresponding quantum states $|b\rangle$). A configuration of the QMSIAM is thus of the form $(\pi, \mathcal{M}, [Q, L])$ where $\pi, \mathcal{M}$ are as in QCSIAM, while $[Q, L]$ is a quantum register.

The structural rules of the QMSIAM are as for the QCSIAM: a single token moves and the quantum register is not updated; instead, the behavior of the QMSIAM differs when the tokens travel through a quantum gate $U \in \mathbb{Q}$ or through an if-then-else. In the first case, illustrated in Figure 9a, the register is updated by applying the gate U to the quantum state. Observe that, when $U = \mathtt{meas}$, the update is inherently probabilistic: the machine actually performs a measurement on its quantum register. In the case of a term $\Gamma, \Delta \vdash \mathtt{if}\ N\ \mathtt{then}\ P_0\ \mathtt{else}\ P_1 : A$, illustrated in Figure 9b, we have two kinds of rules: firstly, we have structural rules allowing tokens to move upwards through the derivation of N; in this way a token may end up in the positive position $N : \mathtt{bit}$; when this happens, the register will contain a value $b \in \{0, 1\}$ (actually, the qubit $|b\rangle$ since we encode bits in the quantum register Q). We then have a second class of rules that allows a token in a negative position in either Δ or A to move upwards within the sub-derivation of P_b: this may only happen once the Boolean b has been determined. The two rules are illustrated in Figure 9. Observe that, in any execution, only one among the sub-derivations of P_0 and P_1 are actually visited by the tokens. Which one may depend on the execution itself (as the Boolean b produced might depend on previous measurements).

In an initial state for the QMSIAM one consider a quantum register $[Q, L]$. By considering *all* possible executions of the machine over $[Q, L]$, we obtain a finite distribution $\mathrm{eval}_{\mathrm{MSIAM}}[Q, L, M] = \sum_{i=1}^{k} p_i Q_i$ of quantum states produced in output by the machine. Via the very similar MSIAM machine, it is not difficult to show that the distribution



(a) Circuit rule.



(b) If-then-else rules.

Figure 9 Rules of QMSIAM.

$\mathrm{eval}_{\mathrm{MSIAM}}[Q, L, M]$ perfectly matches the distribution $\mathrm{eval}_{\lambda}[Q, L, M]$ produced by the λ^{Q}-evaluation of $[Q, L, M]$. In this way, we are reduced to the problem of defining a *simulation* between the QMSIAM and the QCSIAM. This is provided by the following lemma.

Let $\mathcal{C} = (\pi, \mathcal{M}, \Delta_1 \rhd C \rhd \Delta_2)$ be a configuration of the QCSIAM and $\mu = \sum_{i=1}^{k} p_i \mathcal{C}_i$, where $\mathcal{C}_i = (\pi_i, \mathcal{M}_i, [Q_i, L_i])$ be a distribution of configurations for the QMSIAM. For all quantum state $[Q, L]$, let us write $\mu \; S^{[Q,L]} \; \mathcal{C}$ when μ and $\mathcal{C}$ are related as follows:

- in both $\mathcal{C}$ and all $\mathcal{C}_i$ the tokens are in the same positions (i.e. $\mathcal{M} = \mathcal{M}_i$),
- the application of the completely positive map $[\![C]\!]$ to the register $[Q, L]$ produces the *same* distribution of quantum states as μ, that is, $[\![C]\!](\mathrm{state}(Q, L)) = \mathrm{state}(\sum_i p_i Q_i, L)$.

We then have the following result:

▶ **Lemma 12** (the QMSIAM simulates the QCSIAM). *Let $\mathcal{C}, \mathcal{C}'$ be configurations of the QCSIAM and μ be a distribution of configurations for the QMSIAM. If $\mu \; S^{[Q,L]} \; \mathcal{C}$ and $\mathcal{C} \to \mathcal{C}'$ then there exists a distribution μ' such that $\mu' \; S^{[Q,L]} \; \mathcal{C}'$ and such that multiple parallel executions of the QMSIAM lead from μ to μ'.*

From the simulation result above, one can easily obtain a proof of Theorem 11 by induction on a (always terminating) execution of the QCSIAM.

7 Extending the Type System

In this section, we informally describe some additional results about the compilation scheme we introduced and proved correct in this paper.

On the one hand, it should certainly be noted that the type system considered in this work is purely linear, not allowing for any form of duplication. Furthermore, the types are multiplicative, namely $\otimes$ and $\multimap$, and additives are present in disguise only through the type `bit`. Considering a more general type system with additive and exponential connectives can be done, but requires great care. On the one hand, it is in fact well known that Girard's Geometry of Interaction is robust enough to allow for a faithful interpretation of additives and exponentials [21, 33], even in the presence of multiple tokens [11]. On the other hand, the way conditionals are managed here and the special role they have means that the typing of both branches of any conditional needs to be restricted. Any such type should uniquely determine the (finite) *number* and the *shape* of the tokens entering and exiting the conditional. This of course cannot hold in presence of additives, and requires exponentials to be *bounded*, in the style of graded comonads [22]. Similarly, a feature that can be added to our type system without too much trouble is a form of *structural recursion* or *iteration*. However, this

would require the introduction of an inductive type, such as that of natural numbers. The question then becomes the following: should such a type be treated similarly to `bit`, that is, does it indicate the value that can travel on a wire of the constructed circuit, or must its value be known at compile time? In the second case, which we think corresponds to the use of natural numbers in quantum algorithms, it is clear that such an inductive type should itself not occur in the type of the branches of a conditional construct. Summing up, while for the sake of simplicity we have considered a very simple type system here, this does not mean that the approach cannot be adapted to more expressive type systems. The only proviso is that of taking good care of how conditionals can be typed, in particular guaranteeing finiteness and determinacy.

8 Related Work

The literature offers an extensive body of work on quantum compilation, including optimization and so-called transpilation techniques. Techniques specifically tailored to the compilation of higher-order functional languages to quantum circuits are much sparser. One notable contribution in this direction is the language Qunity [44], which offers a higher-order quantum programming language with classical control together with a full compilation scheme towards OpenQASM [10]. However, Qunity does not feature a rewriting system and hence it cannot be executed. This greatly limits the higher-level reasoning that can be done on this language.

Girard's Geometry of Interaction [20, 19] has already been applied to quantum computing [27, 11]. In all the aforementioned work, however, the underlying machine follows the QRAM model, i.e., the token(s) perform some classical computation while moving around the program, from time to time interacting with a quantum register. The token trajectories are *probabilistic*, and provide an alternative way to fully execute the term on a quantum input. By contrast, our approach is fully deterministic, and, instead of executing the term, it produces a (yet to be executed) quantum circuit, thus anticipating the quantum work.

Quantum λ-calculi come in many flavours [41, 40, 2, 12], and semantics frameworks modelling them can themselves be built following distinct lines [40, 37, 7]. One should also mention the extensive body of literature on quantum circuit description languages [38, 25]. The kind of challenges we faced here when compiling conditionals are in fact similar to those encountered when endowing Quipper with so-called dynamic lifting [34, 8].

9 Conclusion

We introduced a compilation scheme turning any term in a linear quantum λ-calculus into an equivalent quantum circuit. The translation is proved correct, and is shown not to give rise to any exponential blowup. Topics for future work include the generalization of the introduced compilation procedure to more expressive calculi and its implementation in concrete programming languages, e.g., Linear Haskell.

References

1 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The machinery of interaction. In *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming*, PPDP '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3414080.3414108.

2 Thorsten Altenkirch and Jonathan Grattage. A functional quantum programming language. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago,*

IL, USA, Proceedings, pages 249–258. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.1.

3 Pablo Arrighi, Alejandro Díaz-Caro, and Benoît Valiron. The vectorial lambda-calculus. *Information and Computation*, 254:105–139, June 2017. URL: http://dx.doi.org/10.1016/j.ic.2017.04.001, doi:10.1016/j.ic.2017.04.001.

4 Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the λ-calculus. *Theoretical Computer Science*, 142(2):277–297, 1995.

5 Anonymous Authors. The geometry of quantum compilation. (supplementary material).

6 Kostia Chardonnet. *Towards a Curry-Howard Correspondence for Quantum Computation*. Theses, Université Paris-Saclay, January 2023. URL: https://theses.hal.science/tel-03959403.

7 Pierre Clairambault and Marc de Visme. Full abstraction for the quantum lambda-calculus. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi:10.1145/3371131.

8 Andrea Colledan and Ugo Dal Lago. On Dynamic Lifting and Effect Typing in Circuit Description Languages. In Delia Kesner and Pierre-Marie Pédrot, editors, *28th International Conference on Types for Proofs and Programs (TYPES 2022)*, volume 269 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2022.3, doi:10.4230/LIPIcs.TYPES.2022.3.

9 Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. Openqasm 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3):1–50, September 2022. URL: http://dx.doi.org/10.1145/3505636, doi:10.1145/3505636.

10 Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language, 2017. URL: https://arxiv.org/abs/1707.03429, arXiv:1707.03429.

11 Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: classical, probabilistic, and quantum effects. *SIGPLAN Not.*, 52(1):833–845, jan 2017. doi:10.1145/3093333.3009859.

12 Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. On a measurement-free quantum lambda calculus with classical control. *Math. Struct. Comput. Sci.*, 19(2):297–335, 2009. doi:10.1017/S096012950800741X.

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

14 Alejandro Díaz-Caro and Gilles Dowek. A linear linear lambda-calculus. *Mathematical Structures in Computer Science*, pages 1–35, May 2024. URL: http://dx.doi.org/10.1017/S0960129524000197, doi:10.1017/s0960129524000197.

15 Yuan Feng and Mingsheng Ying. Quantum hoare logic with classical variables. *ACM Transactions on Quantum Computing*, 2(4):1–43, 2021.

16 Peng Fu, Kohei Kishida, Neil J. Ross, and Peter Selinger. Proto-quipper with dynamic lifting. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. doi:10.1145/3571204.

17 Dan R. Ghica. Geometry of synthesis: a structured approach to vlsi design. *SIGPLAN Not.*, 42(1):363–375, jan 2007. doi:10.1145/1190215.1190269.

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

19 Jean-Yves Girard. Geometry of interaction I: interpretation of System F. In *Studies in Logic and the Foundations of Mathematics*, volume 127, pages 221–260. Elsevier, 1989.

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

21 Jean-Yves Girard. Geometry of interaction iii: accommodating the additives. In *Proceedings of the Workshop on Advances in Linear Logic*, pages 329–389, USA, 1995. Cambridge University Press.

22 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97(1):1–66, 1992. doi:https://doi.org/10.1016/0304-3975(92)90386-T.

23 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. *An Introduction to Quantum Programming in Quipper*, pages 110–124. Springer Berlin Heidelberg, 2013. URL: http://dx.doi.org/10.1007/978-3-642-38986-3_10, doi:10.1007/978-3-642-38986-3_10.

24 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. *ACM SIGPLAN Notices*, 48(6):333–342, June 2013. URL: http://dx.doi.org/10.1145/2499370.2462177, doi:10.1145/2499370.2462177.

25 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 333–342, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2491956.2462177.

26 Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

27 Ichiro Hasuo and Naohiko Hoshino. Semantics of higher-order quantum computation via geometry of interaction. *Ann. Pure Appl. Log.*, 168(2):404–469, 2017. doi:10.1016/J.APAL.2016.10.010.

28 Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with Qiskit, 2024. arXiv:2405.08810, doi:10.48550/arXiv.2405.08810.

29 M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011. doi:10.1038/nature10012.

30 A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003. doi:https://doi.org/10.1016/S0003-4916(02)00018-0.

31 E Knill. Conventions for quantum pseudocode. *Technical report*, 6 1996. URL: https://www.osti.gov/biblio/366453, doi:10.2172/366453.

32 Ugo Dal Lago and Margherita Zorzi. Wave-style token machines and quantum lambda calculi (long version), 2013. URL: https://arxiv.org/abs/1307.0550, arXiv:1307.0550.

33 Olivier Laurent. A token machine for full geometry of interaction (extended abstract). In Samson Abramsky, editor, *Typed Lambda Calculi and Applications*, pages 283–297, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

34 Dongho Lee, Valentin Perrelle, Benoît Valiron, and Zhaowei Xu. Concrete categorical model of a quantum circuit description language with measurement. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FSTTCS.2021.51, doi:10.4230/LIPICS.FSTTCS.2021.51.

35 Louis Lemonnier. The semantics of effects: Centrality, quantum control and reversible recursion, 2024. URL: https://arxiv.org/abs/2406.07216, arXiv:2406.07216.

36 Ian Mackie. The geometry of interaction machine. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 198–208, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/199448.199483.

633 **37** Michele Pagani, Peter Selinger, and Benoît Valiron. Applying quantitative semantics to
634 higher-order quantum computing. *SIGPLAN Not.*, 49(1):647–658, January 2014. doi:10.
635 1145/2578855.2535879.
636 **38** Jennifer Paykin, Robert Rand, and Steve Zdancewic. Qwire: a core language for quantum
637 circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming*
638 *Languages*, POPL '17, pages 846–858, New York, NY, USA, 2017. Association for Computing
639 Machinery. doi:10.1145/3009837.3009894.
640 **39** Peter Selinger. Towards a quantum programming language. *Mathematical. Structures in*
641 *Comp. Sci.*, 14(4):527–586, aug 2004. doi:10.1017/S0960129504004256.
642 **40** Peter Selinger and Benoît Valiron. On a fully abstract model for a quantum linear functional
643 language. *Electronic Notes in Theoretical Computer Science*, 210:123–137, 2008.
644 **41** Peter Selinger and Benoît Valiron. *Quantum Lambda Calculus*, pages 135–172. Cambridge
645 University Press, 2009.
646 **42** Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on
647 a quantum computer. *SIAM review*, 41(2):303–332, 1999.
648 **43** Google AI Quantum Team. Cirq Programming Language. https://quantumai.google/cirq,
649 2018.
650 **44** Finn Voichick, Liyi Li, Robert Rand, and Michael Hicks. Qunity: A unified language for
651 quantum and classical computing. *Proceedings of the ACM on Programming Languages*,
652 7(POPL):921–951, 2023.