FROM LAMBDA CALCULUS TO QUANTUM CIRCUITS THROUGH THE GEOMETRY OF INTERACTION

KOSTIA CHARDONNET [a], UGO DAL LAGO [b], PAOLO PISTONE [c], AND NAOHIKO HOSHINO [d]

[a] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

[b] Department of Computer Science and Engineering, University of Bologna, Italy

[c] Université Claude Bernard Lyon 1

[d] Sojo University, Japan

1. Extended Abstract

Introduction. In the QRAM model of quantum computation [Kni96], a classical computer interacts with a quantum processor by instructing the latter to create new qubits, applying some unitary transformations to the existing qubits or by measuring some of of them. In other words, computation consists in a sequence of exchanges between the classical computer and the quantum processor until an end result (the value of which is intrinsically probabilistic) is obtained. Obviously, the classical computer and the quantum processor do not follow the same rules: while the former is a purely classical device, the latter has to obey the laws of quantum physics. In particular, no quantum data can ever be erased or duplicated and the operations the quantum processor can perform are of a very specific shape.

In order to manipulate this model, several *quantum programming languages* have been developed, from assembly code [CJAA$^+$22], to imperative programming languages with loops and classical tests [FY21] to functional programming language such as the quantum λ-calculus [SV$^+$09]. All of those languages share the same core principle: they manipulate an external quantum memory and can apply quantum operations to it, seen as a black-box. 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. A different family of programming languages [Lem24, DCD24, ADCV17] are instead more focused on what happens *inside* the quantum memory and allow the user to write custom-made quantum operations.

In spite of this profusion, some of the principles underlying such quantum programming languages are in sharp contrast with what happens in practice: quantum architectures are often very hard to be accessed interactively, and require a whole, closed circuit, to be sent to them. The latter is then processed and optimized as a whole. As a result, quantum programs are often written down in minimalistic programming languages for quantum circuits [CJAA$^+$22, FY21], or in circuit description languages like Qiskit or Cirq, which manipulate circuits as ordinary data structures.

The question that motivates our work is thus: would it be possible to compile the QRAM languages, and in particular those with higher-order functions [SV$^+$09], down to quantum circuits? The difficulty of answering this question stems from the inherently interactive nature of the semantics of those languages, as well as from the fact that some of them include *higher-order* functions. More specifically, would it be possible to compile programs in these languages to quantum circuits, getting rid of higher-order features, and also preemptively executing all classical operations?

Key words and phrases: Quantum Computing, Lambda-Calculus, Compilation, Geometry of Interaction.

Preprint submitted to
6th International Workshop on Quantum Compilation

One notable work in this direction is the language Qunity [VLRH23], which offers a higher-order quantum programming language with classical control together with a full compilation scheme towards OpenQASM [CJAA$^+$22]. However, Qunity does not feature a rewriting system and hence it cannot be executed. This limits greatly the higher-level reasoning that can be done on this language.

Contributions. In this work we propose a new compilation procedure for the linear fragment of the quantum lambda calculus [SV$^+$09] onto the quantum circuit model QASM2. Notably, our procedure gets rid of *both* higher-order operations and classical control, and thus provides an effective method to answer the following question: given some well-typed term (with first-order type), what is the underlying circuit that was executed?

The fundamental ingredient of our approach is Girard's *Geometry of Interaction* (GoI) [Gir89b, Gir89a], a method coming from linear logic which can be used to translate a typing derivation for a higher-order λ-calculus into a suitable *abstract machine* [Mac95, Ghi07, ADLV20]. The basic idea of the GoI translation is that, given a type derivation $\Gamma \vdash M : A$ for a term M, one introduces a finite set of tokens which can travel along occurrences of base types in Γ and A throughout the derivation; the paths followed by such tokens give then rise to a sort of circuit relating *positive* and *negative* occurrences of ground types in $\Gamma \vdash M : A$. 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, with inputs and outputs corresponding, respectively to the negative and positive occurrences of the types `bit` and `qbit`. For instance, a type derivation $x : \texttt{qbit}, y : \texttt{qbit} \vdash M : \texttt{qbit} \otimes \texttt{qbit}$ (where M might indeed contain higher-order operations as well as if-then-else control instructions over `bits`) will give rise, after compilation, to a standard quantum circuit with two input qubits and two output qubits.

The framework of GoI has already been applied in the context of quantum computing [DLFVY17], yet in that case 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.

Our compilation procedure works in two steps: first, the typing derivation of the term is translated via the GoI onto a language for quantum circuits with classical control. Then, a second compilation takes place, which translates the circuit onto a QASM2 circuit, notably *eliminating* the use of classical control flow.

In this talk, beyond presenting the compilation procedure, we establish its soundness: we prove that, whenever a term M of the linear quantum lambda-calculus, on a given input state $|\phi\rangle$, produces a state $|\phi'\rangle$ after rewriting, then the circuit produced by compiling M, on input state $|\phi\rangle$, will also evaluate to $|\phi'\rangle$. To prove the soundness for the first part of the compilation procedure we exploit a simulation result with respect to the aforementioned GoI translation of the quantum λ-calculus [DLFVY17]. For the second part, we argue via the standard quantum circuit semantics of *completely positive maps* [Sel04].

References

[ADCV17] 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`.

[ADLV20] 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`.

[CJAA$^+$22] 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, et al. Openqasm 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3):1–50, 2022.

[DCD24] Alejandro Díaz-Caro and Gilles Dowek. A linear linear lambda-calculus. *Mathematical Structures in Computer Science*, page 1–35, May 2024. URL: `http://dx.doi.org/10.1017/S0960129524000197`, `doi:10.1017/s0960129524000197`.

[DLFVY17] 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`.

[FY21] Yuan Feng and Mingsheng Ying. Quantum hoare logic with classical variables. *ACM Transactions on Quantum Computing*, 2(4):1–43, 2021.

[Ghi07] 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`.

[Gir89a] 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.

[Gir89b] Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92(69-108):6, 1989.

[Kni96] E Knill. Conventions for quantum pseudocode. *Technical report*, 6 1996. URL: `https://www.osti.gov/biblio/366453`, `doi:10.2172/366453`.

[Lem24] Louis Lemonnier. The semantics of effects: Centrality, quantum control and reversible recursion, 2024. URL: `https://arxiv.org/abs/2406.07216`, `arXiv:2406.07216`.

[Mac95] Ian Mackie. The geometry of interaction machine. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, page 198–208, New York, NY, USA, 1995. Association for Computing Machinery. `doi:10.1145/199448.199483`.

[Sel04] Peter Selinger. Towards a quantum programming language. *Mathematical. Structures in Comp. Sci.*, 14(4):527–586, aug 2004. `doi:10.1017/S0960129504004256`.

[SV⁺09] Peter Selinger, Benoıt Valiron, et al. Quantum lambda calculus. *Semantic techniques in quantum computation*, pages 135–172, 2009.

[VLRH23] Finn Voichick, Liyi Li, Robert Rand, and Michael Hicks. Qunity: A unified language for quantum and classical computing. *Proceedings of the ACM on Programming Languages*, 7(POPL):921–951, 2023.