Resource-Aware Hybrid Quantum Programming with General Recursion and Quantum Control

Kostia Chardonnet Emmanuel Hainry Romain Péchoux Thomas Vinet

This paper introduces the hybrid quantum language with general recursion `Hyrql`, which is driven towards resource-analysis. By design, `Hyrql` does not require the specification of an initial set of quantum gates. Hence, it is well amenable towards a generic cost analysis, unlike languages that use different sets of quantum gates, which yield quantum circuits of distinct complexity. It also satisfies standard properties of programming languages, such as progress, subject reduction and confluence.

Regarding resource-analysis, we show how to relate the runtime of an expressive fragment of `Hyrql` programs with the size of the corresponding quantum circuits. We also manage to capture the class of functions computable in quantum polynomial time, which, by Yao's Theorem, corresponds to families of circuits of polynomial size. Consequently, this result paves the way for the use of termination and runtime-analysis techniques designed for classical programs to guarantee bounds on the size of quantum circuits.

A complete version is available online at `https://arxiv.org/abs/2510.20452`.

1 Motivations

Most well-known quantum algorithms, such as Shor's algorithm [30], were historically designed on the Quantum Random Access Machine (QRAM) model [21]. In this model, a program interacts with a quantum memory through basic operations complying with the laws of quantum mechanics. These operations include a fixed set of quantum gates, chosen to be universal, as well as a probabilistic measurement of qubits. Consequently, the control flow is purely classical, i.e., depends on the (classical) outcome of a measure. Based on this paradigm, several high-level quantum programming languages have been introduced, each with different purposes and applications, from assembly code [11, 12], to imperative languages [17], circuit description languages [19], and λ-calculi [29].

A relevant issue was to extend these models to programs with *quantum control*, also known as coherent control, enabling a "program as data" treatment for the quantum paradigm. Quantum control consists in the ability to write superposition of programs in addition to superposition of data and increases the expressive power of quantum programming languages. It allows the programmer to write algorithms such as the *quantum switch*, which uses less resource (quantum gates) than algorithms with classical control [10] and is physically implementable [1, 27]. Hence, quantum control provides a computational advantage over classical control [3, 31, 22]. In the last decades, several quantum programming languages implementing this concept have been designed, non-exhaustively [2, 28, 16, 24, 34]. To get the best of both worlds, a natural next step was the development of *hybrid languages*, i.e., languages that allow classical and quantum flow/data to be combined. The hybrid paradigm is conveying considerable practical interest: for example, it is used by quantum variational algorithms, a class of quantum algorithms leveraging both classical and quantum computing resources to find approximate solutions to optimization problems; and the properties of hybrid languages have also been deeply studied, non-exhaustively [32, 33, 13, 7].

Since hybrid languages offer interesting prospects in terms of expressiveness and optimal resource consumption, a relevant and open issue concerns the development of resource-aware (static) analyses

Submitted to:
LIQCS 2026

of their programs. These static analyses can be applied to predict the low-level resources required to execute quantum algorithms, such as the depth or size of a quantum circuit.

2 The Hyrql Programming Language

This work solves the above issue for the first time by introducing a HYbrid Recursive Quantum Language Hyrql on which a resource analysis can be performed. Hyrql is, at its core, a marriage between *pure quantum programming languages* that allow to write down superposition of terms, classical pattern-matching, and the standard λ-calculus. It features the standard constructions of λ-calculus, such as higher-order functions (both linear and non-linear), e.g., the quantum switch, as well as (unbounded) recursion. The quantum aspect is handled by the ability to write down weighted sums of terms, i.e., terms of the shape $\sum_{i=1}^{n} \alpha_i \cdot t_i$ for $\alpha_i \in \mathbb{C}$, which can be equated thanks to an equivalence relation $\equiv$ mimicking the vector space structure. It comes with an operational semantics to define program evaluation, and a type system to ensure programs are well-defined.

To give an intuition of the expressivity of Hyrql, the program bqwalk, defined below, produces all the possible quantum walk paths of size at most n, starting from a quantum state q:

$$\texttt{repeat} \triangleq \texttt{letrec}\, f\, n = \texttt{match}\, n \left\{ 0 \to [\,], S(m) \to |0\rangle :: f\, m \right\}$$

$$\texttt{bqwalk} \triangleq \texttt{letrec}\, f\, q = \lambda n.\texttt{qcase}\, q \left\{ \begin{array}{l} |0\rangle \to |0\rangle :: (\texttt{repeat}\ n) \\ |1\rangle \to |1\rangle :: \texttt{match}\, n \left\{ \begin{array}{c} 0 \to [\,] \\ S(m) \to (f\ |-\rangle\ m) \end{array} \right. \end{array} \right\}$$

This program highlights some of the main features in Hyrql. *Quantum control* is achieved by the qcase construct, which evaluates to either branch depending on the value of q; as q can be a superposition, qcase may evaluate to a superposition of the two branches. Hyrql also allows to define arbitrary terms with a classical structure, e.g., here natural numbers $0, S(m)$ and lists $[\,], h :: t$. Such terms can be used for *classical control* with a match construct. Recursive programs, in particular classical and quantumly controlled, are also elements of Hyrql.

Among the other characteristics of Hyrql, the reduction relation $\rightsquigarrow$ implements a *call-by-value* strategy, and superpositions only reduce when expressed in a *canonical form*, allowing us to avoid reducing terms with a null phase. The type system makes the distinction between *classical* (duplicable and erasable) and *quantum* data which can only be used linearly. It also checks for the orthogonality of terms, which is used to type term superposition. This ensures that any well-typed quantum term is of norm 1.

Proposition 1. Hyrql *satisfies the following standard results of typed programming languages: confluence, subject reduction, and progress.*

One perk of Hyrql is its ability to compile restricted terms to quantum circuits with bounded size; in particular, we have a parsimonious relation between the size and the runtime-complexity of the original term. This is done by restricting our language to a subfragment $\texttt{Hyrql}(T)$, which contains the terms satisfying the following properties:

- A restricted syntax containing only terms that can be represented as a circuit;
- Conditions on recursive calls to get a faithful relation between the runtime of t and the size of the circuit, which is derived from [20];
- Terms must terminate at most in time T.

Note that the above program falls into this restriction, along with other programs that can be encoded in Hyrql, e.g., the quantum switch and the Quantum Fourier Transform (see the complete paper). For this set of programs, we can compile them to circuits, and relate their size with the runtime-complexity of the original term, up to a polynomial overhead.

Theorem 1. *Given a term $t \in \texttt{Hyrql}(T)$, it can be compiled to a family of quantum circuits which approximates t, such that each circuit's size is bounded by $P(T)$, for some polynomial P.*

In particular, when t terminates in polynomial time, the size of the circuit is polynomial; therefore, we can exhibit a characterization of the class of functions that can be computed in quantum polynomial time, known as FBQP [8].

Theorem 2. *The set of terms $\texttt{Hyrql}(\mathrm{Poly})_{\texttt{QList}}$, containing any term $t \in \texttt{Hyrql}(P)$ for some polynomial P and such that t takes as input a qubit list, characterizes exactly* FBQP.

These results rely on obtaining a termination guarantee and runtime complexity certificates on a given term. Such properties have been well-studied over the past decades in Term Rewrite Systems (TRS). Approaches to termination analysis include non-exhaustively recursive path orderings [14, 15], dependency pairs [4], and size-change principle [23]. Similarly, several methods address complexity analysis, e.g., interpretation methods [9, 25, 6], and polynomial path orders [26, 5].

The complete version argues how we can, with a more formal treatment, automatize the search for termination and complexity properties by translating terms of Hyrql into TRS, where the translation preserves termination and complexity bounds. Coming back on the above example, it corresponds to the following term rewrite system:

$$\left\{\begin{array}{cc} \texttt{Repeat}\,0 \to [\,] & \texttt{BQWalk}\,|0\rangle\,m \to |0\rangle :: \texttt{Repeat}\,m \\ \texttt{Repeat}\,S(m) \to |0\rangle :: \texttt{Repeat}\,m & \texttt{BQWalk}\,|1\rangle\,0 \to |1\rangle :: [\,] \\ & \texttt{BQWalk}\,|1\rangle\,S(m) \to |1\rangle :: \texttt{BQWalk}\,(|-\rangle)\,m \end{array}\right\}$$

This translation algorithm is faithful in the sense that one reduction in the TRS world corresponds to a bounded number of reductions in Hyrql. Therefore, any technique applied to this TRS stating termination or runtime-complexity gives back a result in Hyrql, and thus on the quantum circuit.

3 Perspectives and Talk Structure

As discussed, one natural next step is to formally implement this translation from Hyrql to TRS, and to show that it, as expected, implies a relation between termination and runtime-complexity in both worlds. Techniques in the TRS world also need to be adapted to fit the quantum setting (i.e., fit the reduction strategy and superpositions).

While terms can be compiled to quantum circuits, this process could be refined, either to characterize more precise complexity classes, as in [18], or to provide an actual compilation algorithm.

The talk we propose is organized as follows:

- We will start by presenting Hyrql, its syntax, type system and operational semantics through a series of examples to show its expressiveness.
- We will then discuss about the properties of the language, and in particular how we can compile Hyrql to quantum circuits, guarantee a bound on the size of the circuit, and characterize FBQP.
- Finally, we will discuss the hardness of proving termination, and why using tools from TRS is relevant. In particular, we will give the intuition behind how a translation from Hyrql to TRS can be achieved, and why it will allow us to obtain back complexity results in the original language, thus on quantum circuits.

References

[1] Alastair A. Abbott et al. “Communication through coherent control of quantum channels”. In: *Quantum* 4 (Sept. 2020), p. 333. ISSN: 2521-327X. DOI: 10.22331/q-2020-09-24-333.

[2] Thorsten Altenkirch and Jonathan Grattage. “A Functional Quantum Programming Language”. In: *Logic in Computer Science, LICS’05*. IEEE Computer Society, 2005, pp. 249–258. DOI: 10.1109/lics.2005.1.

[3] Mateus Araújo, Fabio Costa, and Časlav Brukner. “Computational Advantage from Quantum-Controlled Ordering of Gates”. In: *Phys. Rev. Lett.* 113 (25 Dec. 2014), p. 250402. DOI: 10.1103/PhysRevLett.113.250402.

[4] Thomas Arts and Jürgen Giesl. “Termination of term rewriting using dependency pairs”. In: *Theoretical Computer Science* 236.1 (2000), pp. 133–178. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(99)00207-8.

[5] Martin Avanzini and Georg Moser. “Dependency pairs and polynomial path orders”. In: *International Conference on Rewriting Techniques and Applications, RTA’09*. Springer. 2009, pp. 48–62. DOI: 10.1007/978-3-642-02348-4_4.

[6] Patrick Baillot and Ugo Dal Lago. “Higher-Order Interpretations and Program Complexity”. In: *Information and Computation* 248 (2016), pp. 56–81. DOI: 10.1016/j.ic.2015.12.008.

[7] Kathleen Barsse, Romain Péchoux, and Simon Perdrix. *A Quantum Programming Language for Coherent Control*. 2025. arXiv: 2507.10466.

[8] Ethan Bernstein and Umesh Vazirani. “Quantum Complexity Theory”. In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1411–1473. ISSN: 0097-5397. DOI: 10.1137/S0097539796300921.

[9] Guillaume Bonfante, Jean-Yves Marion, and Jean-Yves Moyen. “Quasi-Interpretations a Way to Control Resources”. In: *Theoretical Computer Science* 412.25 (June 2011), pp. 2776–2796. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2011.02.007.

[10] Giulio Chiribella et al. “Quantum computations without definite causal structure”. In: *Phys. Rev. A* 88 (2 Aug. 2013), p. 022318. DOI: 10.1103/PhysRevA.88.022318.

[11] Andrew W. Cross et al. *Open Quantum Assembly Language*. 2017. arXiv: 1707.03429 [quant-ph].

[12] Andrew W. Cross et al. “OpenQASM 3: A Broader and Deeper Quantum Assembly Language”. In: *ACM Transactions on Quantum Computing* 3.3 (Sept. 2022), pp. 1–50. ISSN: 2643-6817. DOI: 10.1145/3505636.

[13] Kinnari Dave et al. “Combining quantum and classical control: syntax, semantics and adequacy”. In: *Foundations of Software Science and Computation Structures, FoSSaCS 2025*. Hamilton, ON, Canada: Springer-Verlag, 2025, pp. 155–175. ISBN: 978-3-031-90896-5. DOI: 10.1007/978-3-031-90897-2_8.

[14] Nachum Dershowitz. “Orderings for term-rewriting systems”. In: *Theoretical computer science* 17.3 (1982), pp. 279–301. DOI: 10.1016/0304-3975(82)90026-3.

[15] Nachum Dershowitz. “Termination of rewriting”. In: *J. Symb. Comput.* 3.1-2 (Feb. 1987), pp. 69–115. ISSN: 0747-7171. DOI: 10.1016/S0747-7171(87)80022-6.

[16] Alejandro Díaz-Caro and Octavio Malherbe. “Quantum Control in the Unitary Sphere: Lambda-S1 and its Categorical Model”. In: *Log. Methods Comput. Sci.* 18.3 (2022). DOI: 10.46298/LMCS-18(3:32)2022.

[17] Yuan Feng and Mingsheng Ying. “Quantum Hoare logic with classical variables”. In: *ACM Transactions on Quantum Computing* 2.4 (2021), pp. 1–43. DOI: 10.1145/3456877.

[18] Florent Ferrari et al. “Quantum Programming in Polylogarithmic Time”. In: *Mathematical Foundations of Computer Science, MFCS 2025*. Vol. 345. LIPIcs. 2025, 47:1–47:17. DOI: 10.4230/LIPICS.MFCS.2025.47.

[19] Alexander S. Green et al. “Quipper: a scalable quantum programming language”. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI ’13*. ACM, June 2013, pp. 333–342. DOI: 10.1145/2491956.2462177.

[20] Emmanuel Hainry, Romain Péchoux, and Mário Silva. “Branch Sequentialization in Quantum Polytime”. In: *Formal Structures for Computation and Deduction, FSCD 2025*. Ed. by Maribel Fernández. Vol. 337. LIPIcs. 2025, 22:1–22:22. DOI: 10.4230/LIPICS.FSCD.2025.22.

[21] Emanuel Knill. *Conventions for quantum pseudocode*. Tech. rep. Los Alamos National Lab., 1996. arXiv: 2211.02559.

[22] Hlér Kristjánsson et al. *Exponential separation in quantum query complexity of the quantum switch with respect to simulations with standard quantum circuits*. 2024. arXiv: 2409.18420.

[23] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. “The size-change principle for program termination”. In: *Symposium on Principles of Programming Languages, POPL 2001*. New York, NY, USA: ACM, Jan. 2001, pp. 81–92. DOI: 10.1145/373243.360210.

[24] Louis Lemonnier. “The Semantics of Effects : Centrality, Quantum Control and Reversible Recursion”. PhD thesis. Université Paris-Saclay, June 2024.

[25] Jean-Yves Marion and Romain Péchoux. “Sup-interpretations, a semantic method for static analysis of program resources”. In: *ACM Trans. Comput. Logic* 10.4 (Aug. 2009). ISSN: 1529-3785. DOI: 10.1145/1555746.1555751.

[26] Georg Moser. “Proof Theory at Work: Complexity Analysis of Term Rewrite Systems”. Cumulative Habilitation Thesis. University of Innsbruck, 2009. eprint: 0907.5527.

[27] Lorenzo M. Procopio et al. “Experimental superposition of orders of quantum gates”. In: *Nature Communications* 6.1 (Aug. 2015). ISSN: 2041-1723. DOI: 10.1038/ncomms8913.

[28] Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. “From Symmetric Pattern-Matching to Quantum Control”. In: *Foundations of Software Science and Computation Structures*. Ed. by Christel Baier and Ugo Dal Lago. Cham: Springer International Publishing, 2018, pp. 348–364. ISBN: 978-3-319-89366-2. DOI: 10.1007/978-3-319-89366-2_19.

[29] Peter Selinger and Benoît Valiron. “Semantic Techniques in Quantum Computation”. In: *Quantum lambda calculus*. Ed. by Simon Gay and Ian Mackie. Cambridge University Press, 2009, pp. 135–172. DOI: 10.1017/cbo9781139193313.005.

[30] 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.

[31] Márcio M. Taddei et al. “Computational Advantage from the Quantum Superposition of Multiple Temporal Orders of Photonic Gates”. In: *PRX Quantum* 2 (1 Feb. 2021), p. 010320. DOI: 10.1103/PRXQuantum.2.010320.

[32] Finn Voichick et al. “Qunity: A Unified Language for Quantum and Classical Computing”. In: *Proc. ACM Program. Lang.* 7.POPL (Jan. 2023). DOI: 10.1145/3571225.

[33] Mingsheng Ying. *Foundations of quantum programming*. Elsevier, 2024. DOI: 10.1016/C2014-0-02660-3.

[34] Charles Yuan, Agnes Villanyi, and Michael Carbin. “Quantum Control Machine: The Limits of Control Flow in Quantum Programming”. In: *Proceedings of the ACM on Programming Languages* 8.OOPSLA1 (Apr. 2024), pp. 1–28. ISSN: 2475-1421. DOI: 10.1145/3649811.