A hybrid and reversible quantum language

Kostia CHARDONNET Emmanuel HAINRY Romain PÉCHOUX Thomas VINET

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

We present a typed quantum reversible language that features both classical and quantum data, a minimal but expandable set of types, quantum superpositions and recursion. We introduce a syntax to define reversible terms using a set of clauses and pattern-matching; and while verifying reversibility is undecidable in most cases, we give a decidable predicate to check and find the inverse for a significant subset of reversible terms. Finally, we discuss about viewing our language as a term-rewriting system in order to analyze the resources used by a program.

1 Introduction

In quantum computation, measurement is the only non-reversible operation. Moreover, as measurements can always be delayed, it is natural to consider a purely reversible fragment without measure. Such fragments have been studied through a quantum functional language, with linear types and algebraic terms in order to allow for quantum superpositions. This idea has been achieved with a language inspired from lambda calculus in [2, 10], then a type discipline was given in [1]. From this design, reversibility has been considered in [6] by defining unitary operations and ensuring only norm-1 superpositions. This approach nonetheless features only basic types, and superpositions are restricted in their expressivity. Another approach has been taken in [9], inspired from [7], which uses pattern-matching to define its reversible terms, called *isos*. This language features some basic types, recursion, lists as an higher-order type, and an extension to quantum with term superpositions. This allows to define reversible fragments on infinite dimension spaces. This language has then been refined in [5, 4, 8] to feature additional types, to prove soundness of the definitions and discuss about completeness. However, these refinements drop the quantum aspect, and superpositions are either not present or not normalized. Furthermore, both approaches only feature linear data.

(Contributions) We present a typed quantum reversible language, which is an enhanced version of the language designed in [4] with more expressive power. This language has a hybrid design with classical and quantum control flow, allowing the programmer to build unitaries directly. The type discipline distinguishes between linear and non-linear terms in a standard manner to delineate a clear separation between quantum and classical data. It also takes care of only typing terms that are normalized to emulate a unitary behaviour for iso constructs. We also introduce a *struct* operator, which allows to extract the classical structure of quantum data and to use it non-linearly, and gives the language a hybrid behaviour by computing classical properties of quantum data, for example, the length of a qubit list. This language is equipped with a rewriting relation, on which we prove standard results like subject reduction. The iso contructs of [4] define reversible computation and are given a special type for reversibility. Our notion of isos is strictly more general but reversibility is undecidable. However, it can be restricted to obtain a decidable criterion, similarly to [8].

This paper is structured as follows. We first introduce the syntax of the language, with its type system and its operational semantics. We then show different examples of isos to illustrate the expressivity of

Submitted to:
QPL 2025

the language, and we give the results obtained so far. Finally, we discuss about future work, and how to control the resources used by a given program.

2 Language

$$
\begin{array}{lll}
\text{(Classical types)} & C & ::= B_C \mid C \Rightarrow T \mid T \multimap T \mid B \leftrightarrow B \\
\text{(Terms types)} & T & ::= B_Q \mid C \\
\text{(Values)} & v & ::= x \mid \alpha v \mid \alpha v_1 + \beta v_2 \mid c(v_1,\dots,v_n) \mid \{v_1 \leftrightarrow t_1 \mid \dots\} \mid \lambda x.v \\
\text{(Terms)} & t & ::= x \mid \alpha t \mid \alpha t_1 + \beta t_2 \mid c(t_1,\dots,t_n) \mid \texttt{let}\ (x_1,\dots,x_n) = t_1\ \texttt{in}\ t_2 \\
 & & \quad \mid \texttt{struct}(t) \mid \{v_1 \leftrightarrow t_1 \mid \dots\} \mid \texttt{fix}\ f.t \mid \lambda x.t \mid t_1 t_2
\end{array}
$$

Table 1: Grammar and types for terms and values.

(Types) The language contains built types, which represent types of order 0. Each built type $A, B \in \mathbb{B}$ comes with its set of constructors $\mathrm{Cons}(B) = \{c_1,\dots,c_n\}$. We use the notation $c :: \times_{i=1}^{n} B_i \to B$ for $c \in \mathrm{Cons}(B)$ to indicate its input types. Built types can be seen as the disjoint union of quantum, $B_Q \in \mathbb{B}_Q$, and classical, $B_C \in \mathbb{B}_C$, built types. This allows us to define some standard types, for example the unit type $\mathbb{1}$ with a single constructor $() :: \mathbb{1}$. We can also define tensor and cartesian product for given built types A, B naturally. Inductive types can also be defined; for instance, natural numbers `nat` are defined with two constructors: $\mathrm{Cons}(\texttt{nat}) = \{0, S\}$, with $0 :: \texttt{nat}$, $S :: \texttt{nat} \to \texttt{nat}$. Lists are defined in the same fashion, with an empty list constructor $[\,]$ and a head-tail constructor $h :: t$. We can then define classical types, which contain classical built types, iso constructs, linear and non-linear abstraction. General types extend classical types by including quantum built types.

(Grammar) The grammar of the language is defined in Table 1. The language contains base elements of lambda calculus, namely variables in a countable set, abstractions, and term applications. Terms can be wrapped under a constructor c; the `let` construct can destruct n-uplets and is also useful to do intermediate computation. Given a quantum term t, $\texttt{struct}(t)$ extracts its structure to use it classically: for a list of qubits $t = |0\rangle :: |1\rangle :: [\,]$, $\texttt{struct}(t)$ is a classical list of same size, with no information about the values of the qubits. Any quantum term can be superposed, and the type discipline ensures normalization; we add an equivalence relation $\equiv$ on superpositions, shown in the appendix, such that we have linearity for isos and constructors, and we can define the summation symbol $\sum$ unambiguously. This allows us to define pure terms as terms that cannot be expanded linearly under this relation; we can then give each term an unique canonical form $\sum_i \alpha_i t_i \in \texttt{CAN}$. Finally, an iso construct, written ω, is made of multiple clauses between values and terms, on which we add constraints below to obtain reversibility.

$$
\frac{\Gamma;\Delta_i \vdash t_i : B_Q \qquad \forall i \neq j,\ t_i \perp t_j \qquad \sum_{i=1}^{n} |\alpha_i|^2 = 1}{\Gamma;\Delta_1,\dots,\Delta_n \vdash \sum_{i=1}^{n} \alpha_i t_i : B_Q}
\qquad
\frac{\Gamma;\Delta \vdash t_1 : C \Rightarrow T \qquad \Gamma;\varnothing \vdash t_2 : C}{\Gamma;\Delta \vdash t_1 t_2 : T}
$$

$$
\frac{\Gamma;\Delta_i \vdash v_i : A \qquad \Gamma;\Delta_i \vdash t_i : B \qquad \texttt{iso}(\{v_1 \leftrightarrow t_1 \mid \dots\})}{\Gamma;\varnothing \vdash \{v_1 \leftrightarrow t_1 \mid \dots\} : A \leftrightarrow B}
\qquad
\frac{\Gamma, x : C;\Delta \vdash t : T}{\Gamma;\Delta \vdash \lambda x.t : C \Rightarrow T}
\qquad
\frac{\Gamma;\Delta, x : T \vdash t : T'}{\Gamma;\Delta \vdash \lambda x.t : T \multimap T'}
$$

Table 2: Excerpt of the typing rules for terms

(Typing rules) We present a glimpse of the typing rules in Table 2. Due to space limitation, we present only the most important rules here, but all of them are available in the appendix. The typing

context $\Gamma;\Delta$ has a linear typing discipline and is split in two: a non-linear context Γ made only of classical variables $x : C$; and a linear context Δ which can contain any variable. As we want well-typed terms to be normalized, the superposition's typing requires that terms are two by two orthogonal and phases satisfy a normalization condition. Orthogonality is given by the predicate $\perp$ defined in the appendix on $\mathbb{B}_Q$. Finally, we impose a semantic definition $\texttt{iso}(\omega)$ that requires isos to be reversible and also that any pure value will match uniquely one or a superposition of branches of a given iso.

$$\frac{t \to t'}{C_E(t) \to C_E(t')} \qquad \frac{}{\texttt{fix}\ f.\omega \to \omega[f/\texttt{fix}\ f.\omega]} \qquad \frac{}{(\lambda x.t)v \to t[x/v]} \qquad \frac{\sigma(p) = v}{\texttt{let}\ p = v\ \texttt{in}\ t \to \sigma(t)}$$

$$\frac{v = \sum_i \alpha_i \sigma_i(v_i) \qquad \texttt{FV}(v) = \emptyset}{\{v_1 \leftrightarrow t_1 \ \mid\ \ldots\}v \to \sum_i \alpha_i \sigma_i(t_i)} \qquad \frac{t \to t'}{t \rightsquigarrow t'} \qquad \frac{t \equiv \sum_i \alpha_i t_i \in \texttt{CAN} \qquad t_i \xrightarrow{=} t_i' \qquad \exists i_0,\ t_i \to t_i'}{t \rightsquigarrow \sum_i \alpha_i t_i'}$$

Table 3: Excerpt of the semantic reduction rules

(Operational semantics) The language comes with a general reduction relation $\rightsquigarrow$ acting on terms which follows a call-by-value strategy. We again present some rules in Table 3, and the exhaustive set of rules is in the appendix. Reduction is first defined on pure terms with $\to$, then is extended by reducing each term of the canonical form when t is not a value. C_E represents an evaluation context, defined in the appendix; σ is a partial map from variables to values and called a substitution.

3 Examples and results

In this language, we are able to type lists superpositions, for example $\frac{1}{\sqrt{2}}(|0\rangle :: |0\rangle :: [\,] + |1\rangle :: |1\rangle :: [\,])$. Isos can represent various transformations: the Hadamard matrix, a map function acting on each element of a list, measuring the length of a quantum list, etc. It can also simulate quantum control.

$$\begin{aligned}
\texttt{Had} &= \{|0\rangle \leftrightarrow \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \ \mid\ |1\rangle \leftrightarrow \frac{1}{\sqrt{2}}|1\rangle - \frac{1}{\sqrt{2}}|0\rangle\} \\
\texttt{map} &= \lambda\phi.\texttt{fix}\ f.\{[\,] \leftrightarrow [\,] \mid h :: t \leftrightarrow \phi h :: ft\} \\
\texttt{len}\ t &= (\texttt{fix}\ f.\{[\,] \leftrightarrow 0 \mid h :: t \leftrightarrow S\ f\ t\})\texttt{struct}(t) \\
\texttt{QSwitch} &= \lambda f.\lambda g.\{|0\rangle \otimes q \leftrightarrow |0\rangle \otimes f(gq) \mid |1\rangle \otimes q \leftrightarrow |1\rangle \otimes g(fq)\}
\end{aligned}$$

Property (Subject reduction)**.** Let $\Gamma;\Delta \vdash t : T$ a well-typed term. If $t \rightsquigarrow t'$, then $\Gamma;\Delta \vdash t' : T$.

Lemma. Let $\omega = \{v_i \leftrightarrow e_i\}$, where e_i is a succession of `let` constructors. Then we have a predicate on ω that can verify reversibility and exhibits the inverse iso.

4 Future work

We have introduced a hybrid and reversible quantum language, that features both classical and quantum terms, with more expressive power. We intend to view our language (especially isos) as a term rewriting system. Using this, we could use some known techniques in order to guarantee some complexity bounds on the resources used for a given program. One technique we can use on term-rewriting sytems to achieve this is to use quasi-interpretations [3]. This allows us to have more control on the resources that a given program uses, for example the number of gates used, the depth of a circuit, etc.

References

[1] Pablo Arrighi, Alejandro Díaz-Caro & Benoît Valiron (2017): *The Vectorial λ-Calculus*. Information and Computation 254, pp. 105–139, doi:10.1016/j.ic.2017.04.001.

[2] Pablo Arrighi & Gilles Dowek (2017): *Lineal: A Linear-Algebraic Lambda-calculus*. Logical Methods in Computer Science Volume 13, Issue 1, doi:10.23638/LMCS-13(1:8)2017.

[3] G. Bonfante, J. Y. Marion & J. Y. Moyen (2011): *Quasi-Interpretations a Way to Control Resources*. Theoretical Computer Science 412(25), pp. 2776–2796, doi:10.1016/j.tcs.2011.02.007.

[4] Kostia Chardonnet, Louis Lemonnier & Benoît Valiron (2024): *Semantics for a Turing-Complete Reversible Programming Language with Inductive Types*. In Jakob Rehof, editor: 9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024), Leibniz International Proceedings in Informatics (LIPIcs) 299, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 19:1–19:19, doi:10.4230/LIPIcs.FSCD.2024.19.

[5] Kostia Chardonnet, Alexis Saurin & Benoît Valiron (2023): *A Curry-Howard Correspondence for Linear, Reversible Computation*. LIPIcs, Volume 252, CSL 2023 252, pp. 13:1–13:18, doi:10.4230/LIPICS.CSL.2023.13.

[6] Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel & Benoît Valiron (2019): *Realizability in the Unitary Sphere*. In: 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 1–13, doi:10.1109/LICS.2019.8785834. arXiv:1904.08785.

[7] Roshan P. James & Amr Sabry (2012): *Information Effects*. In: Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Popl '12, Association for Computing Machinery, New York, NY, USA, pp. 73–84, doi:10.1145/2103656.2103667.

[8] Louis Lemonnier (2024): *The Semantics of Effects : Centrality, Quantum Control and Reversible Recursion*. Ph.D. thesis, Université Paris-Saclay.

[9] Amr Sabry, Benoît Valiron & Juliana Kaizer Vizzotto (2018): *From Symmetric Pattern-Matching to Quantum Control*. In Christel Baier & Ugo Dal Lago, editors: Foundations of Software Science and Computation Structures, Springer International Publishing, Cham, pp. 348–364, doi:10.1007/978-3-319-89366-2_19.

[10] Lionel Vaux (2009): *The Algebraic Lambda Calculus*. Mathematical Structures in Computer Science 19(5), pp. 1029–1059, doi:10.1017/S0960129509990089.

A Appendix

We define the equivalence relation $\equiv$ between terms in Table 4. Given a term t, we can find its unique canonical form given as $t = \sum_i \alpha_i t_i$, where $\alpha_i \neq 0$, $t_i \neq t_j$, and t_i are pure terms, i.e. cannot be developed under the last two rules.

$$t_1 + t_2 \equiv t_2 + t_1 \qquad t_1 + (t_2 + t_3) \equiv (t_1 + t_2) + t_3 \qquad 1 \cdot t \equiv t \qquad t + 0 \cdot t' \equiv t$$
$$\alpha \cdot (\beta \cdot t) \equiv \alpha\beta \cdot t \qquad \alpha \cdot (t_1 + t_2) \equiv \alpha \cdot t_1 + \alpha \cdot t_2 \qquad (\alpha + \beta) \cdot t \equiv \alpha \cdot t + \beta \cdot t$$
$$\omega(\textstyle\sum \alpha_i \cdot t_i) \equiv \sum \alpha_i \cdot \omega t_i \qquad c(t_1, \ldots, \sum_i \alpha_i \cdot t^i_j, \ldots, t_m) \equiv \sum_i \alpha_i \cdot c(t_1, \ldots, t^i_j, \ldots, t_m)$$

Table 4: Equivalence relation for superposition of quantum terms

The following tables contain the definition of the orthogonality predicate $\perp$, and the typing rules. The notation $[x : T]$ means that this context is optional. The type F represents an iso-like type: $F ::= A \leftrightarrow B \mid T \multimap F \mid C \Rightarrow F$.

$$\frac{i \neq j}{|i\rangle \perp |j\rangle} \qquad \frac{v \perp v'}{v \otimes q \perp v' \otimes q'} \qquad \frac{v \perp v'}{q \otimes v \perp q' \otimes v'} \qquad \frac{\forall i \neq j, v_i \perp v_j \qquad \sum_{i=1}^{d} \alpha_i \alpha_i'^{*} = 0}{\sum \alpha_i v_i \perp \sum \alpha_i' v_i}$$

$$\frac{t \overset{*}{\rightsquigarrow} v \qquad t' \overset{*}{\rightsquigarrow} v' \qquad v \perp v'}{t \perp t'}$$

Table 5: Rules for the orthogonality of quantum states.

$$\frac{}{\Gamma; x : T \vdash x : T} \qquad \frac{}{\Gamma, x : C; \varnothing \vdash x : C} \qquad \frac{\Gamma; \Delta_i \vdash t_i : B_i \qquad c :: \times_{i=1}^{n} B_i \to B}{\Gamma; \Delta_1, \ldots, \Delta_n \vdash c(t_1, \ldots, t_n) : B}$$

$$\frac{\Gamma; \Delta_i \vdash t_i : B_Q \qquad \forall i \neq j,\ t_i \perp t_j \qquad \sum_{i=1}^{n} |\alpha_i|^2 = 1}{\Gamma; \Delta_1, \ldots, \Delta_n \vdash \sum_{i=1}^{n} \alpha_i t_i : B_Q}$$

$$\frac{\Gamma; \Delta \vdash t : \times_{i=1}^{n} A_i \qquad \Gamma; \Delta', x_1 : A_1, \ldots, x_n : A_n \vdash t' : B}{\Gamma; \Delta, \Delta' \vdash \texttt{let}\ (x_1, \ldots, x_n) = t\ \texttt{in}\ t' : B} \qquad \frac{\Gamma; \Delta \vdash t : B}{\Gamma, \texttt{struct}(t) : \diamond B; \varnothing \vdash \texttt{struct}(t) : \diamond B}$$

$$\frac{\Gamma; \Delta_i \vdash v_i : A \qquad \Gamma; \Delta_i \vdash t_i : B \qquad \texttt{iso}(\{v_1 \leftrightarrow t_1 \mid \ldots\})}{\Gamma; \varnothing \vdash \{v_1 \leftrightarrow t_1 \mid \ldots\} : A \leftrightarrow B} \qquad \frac{\Gamma, f : F \vdash \omega : F}{\Gamma; \varnothing \vdash \texttt{fix}\ f.\omega : F}$$

$$\frac{\Gamma, x : C; \Delta \vdash t : T}{\Gamma; \Delta \vdash \lambda x.t : C \Rightarrow T} \qquad \frac{\Gamma; \Delta, x : T \vdash t : T'}{\Gamma; \Delta \vdash \lambda x.t : T \multimap T'}$$

$$\frac{\Gamma; \Delta \vdash t_1 : C \Rightarrow T \qquad \Gamma; \varnothing \vdash t_2 : C}{\Gamma; \Delta \vdash t_1 t_2 : T} \qquad \frac{\Gamma, [\texttt{struct}(t_2) : \diamond T]; \Delta \vdash t_1 : T \multimap T' \qquad \Gamma; \Delta' \vdash t_2 : T}{\Gamma; \Delta, \Delta' \vdash t_1 t_2 : T'}$$

$$\frac{\Gamma; \varnothing \vdash \omega : A \leftrightarrow B \qquad \Gamma; \Delta \vdash t : A}{\Gamma; \Delta \vdash \omega t : B}$$

Table 6: Complete typing rules for terms

The table below contains all the reduction rules. $\overset{=}{\to}$ denotes the reflexive closure, meaning there is either no reduction step or one reduction step. The evaluation context is defined formally as:

$$C_E ::= [\,] \mid c(v_1, \ldots C_E, t_m, \ldots) \mid \texttt{let}\ p = C_E\ \texttt{in}\ t \mid \texttt{let}\ p = v\ \texttt{in}\ C_E \mid \texttt{struct}(C_E)$$

where v, v_1 are values, t, t_m are terms, and c is a constructor.

The table below contains the predicate OD that is used to prove reversibility. This predicate is defined only for values of order 0. We split constructors as $\mathrm{Cons}(B) = \mathrm{Cons}(B)^{+} \cup \mathrm{Cons}(B)^{0}$, meaning constructors with at least one input and constructors with no input.

$$\frac{t \to t'}{C_E(t) \to C_E(t')} \qquad \frac{}{\texttt{fix}\ f.\omega \to \omega[f/\texttt{fix}\ f.\omega]} \qquad \frac{}{(\lambda x.t)v \to t[x/v]} \qquad \frac{\sigma(p) = v}{\texttt{let}\ p = v\ \texttt{in}\ t \to \sigma(t)}$$

$$\frac{v : Q_d}{\texttt{struct}(v) \to d} \qquad \frac{\Gamma;\varnothing \vdash v : B_C}{\texttt{struct}(v) \to v} \qquad \frac{c :: B_1 \times \ldots B_n \to B \qquad \Gamma;\Delta_i \vdash v_i : B_i \qquad B \in \mathbb{B}_Q,\ B \neq Q_d}{\texttt{struct}(c(v_1,\ldots,v_n)) \to \tilde{c}(\texttt{struct}(v_1),\ldots,\texttt{struct}(v_n))}$$

$$\frac{v = \sum_i \alpha_i \sigma_i(v_i) \qquad \texttt{FV}(v) = \emptyset}{\{v_1 \leftrightarrow t_1 \ \mid\ \ldots\} v \to \sum_i \alpha_i \sigma_i(t_i)}$$

$$\frac{t \to t'}{t \rightsquigarrow t'} \qquad \frac{}{\texttt{struct}(\alpha v_1 + \beta v_2) \rightsquigarrow \texttt{struct}(v_1)} \qquad \frac{t \equiv \sum_i \alpha_i t_i \in \texttt{CAN} \qquad t_i \stackrel{=}{\to} t_i' \qquad \exists i_0,\ t_i \to t_i'}{t \rightsquigarrow \sum_i \alpha_i t_i'}$$

Table 7: Complete semantic reduction rules

$$\frac{}{\texttt{OD}_A(\{x\})} \qquad \frac{\texttt{OD}_A(\pi_1(S)) \qquad \forall a \in \pi_1(S), \texttt{OD}_B(S_a^1)}{\texttt{OD}_{A\times B}(S)} \qquad \frac{\texttt{OD}_A(\pi_2(S)) \qquad \forall b \in \pi_2(S), \texttt{OD}_B(S_b^2)}{\texttt{OD}_{A\times B}(S)}$$

$$\frac{\texttt{OD}_A(S) \qquad (\alpha_{e,v})_{(e,v)\in S\times S} \text{ is a unitary matrix}}{\texttt{OD}_A(\{\sum_{v\in S} \alpha_{e,v} v \mid e \in S\})} \qquad \frac{\forall c_i :: \vec{B_i} \to B \in \text{Cons}(B)^+, \texttt{OD}_{\vec{B_i}}(S_i)}{\texttt{OD}_B((\cup_{c\in \text{Cons}(B)^0} c) \cup (\cup_{c_i \in \text{Cons}(B)^+} c_i(S_i)))}$$

Table 8: Definition of OD