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

Kostia Chardonnet$^{1[0009-0000-0671-6390]}$, Emmanuel Hainry$^{1[0000-0002-9750-0460]}$, Romain Péchoux$^{1[0000-0003-0601-5425]}$, and Thomas Vinet$^{1[0009-0007-8547-6145]}$

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

Abstract. This paper introduces the hybrid quantum language with general recursion `Hyrql`, driven towards resource-analysis. By design, `Hyrql` does not require the specification of an initial set of quantum gates and, hence, is well amenable towards a generic cost analysis. Indeed, languages using different sets of quantum gates lead to representations of quantum circuits whose complexity varies. Towards resource-analysis, a semantics-preserving compilation algorithm to simply-typed term rewrite systems is described; allowing a generic reuse of all known techniques for analyzing the complexity of term rewrite systems. We prove the versatility of this approach through many examples.

Keywords: Hybrid Quantum Programs, Resource Analysis, Rewrite Systems

1 Introduction

Motivations. Most well-known quantum algorithms, such as Shor's algorithm [49], were historically designed on the QRAM model [33]. 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 [20,19], to imperative languages [27], tools for formal verification [16], circuit description languages [29], and λ-calculi [48].

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 superpositions of programs in addition to superpositions 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 fewer resources (quantum gates) than algorithms with classical control [17] and is physically implementable [1,43]. Hence, quantum control provides a computational advantage over classical control [3,50,34]. In the last decades, several quantum programming languages implementing this concept have been designed, non-exhaustively [2,44,25,38]. 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. Such languages have also been deeply studied [51,30,53,22,10]. This keen interest is a direct consequence of the fact that hybrid languages can be used to design variational quantum algorithms, a class of quantum algorithms leveraging both classical and quantum computing resources to find approximate solutions to optimization problems.

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 of their programs. These static analyses could be applied to highlight and clarify the advantage of variational algorithms over their QRAM-based counterparts.

Contributions. This paper 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 a hybrid extension of the functional quantum language `Spm` (*Symmetrical pattern-matching*) of [44,38]. In `Spm`, the programmer can write down directly unitary applications, combining quantum superpositions of terms with pattern-matching. Unitarity is enforced by a (decidable) linear typing discipline using an orthogonality predicate. `Spm` is, however, a purely quantum language, which makes the manipulation (and, consequently, resource analysis) of classical information awkward: they can be neither discarded, nor duplicated by linearity. The hybrid nature of `Hyrql` relies on a typing discipline which delineate a clear separation between quantum/linear and classical/non-linear data in the typing contexts. Similarly to `Spm`, `Hyrql` does not include measurement: the flow from quantum data to classical data is handled through the use of a `shape` construct, as introduced in [29], which returns the classic structural information on data containing quantum information, without getting any information on the value of the quantum states. For example, the `shape` of a qubit list provides classical (duplicable) information, such as the number of qubits. The decision to provide `Hyrql` with a `Spm`-architecture was motivated by two important factors. First, in contrast with most of its competitors, `Spm` does not require the inclusion of an initial set of quantum gates, making its resource analysis generic. Second, thanks to its pattern-matching design, `Spm` allows for the reuse of a wide variety of tools designed for resource analysis (e.g., termination or complexity) of term rewriting systems.

Our paper contains the following main contributions:

– The introduction of the hybrid language `Hyrql` with general recursion as well as its operational semantics, type system, and standard properties: subject reduction (Lemma 4), progress (Lemma 3), and confluence (Theorem 1).
– A proof that the orthogonality predicate in `Hyrql` is Π_2^0-complete, hence undecidable (Theorem 2). Consequently, typing is not decidable. This is not surprising as the language encompasses general recursion. The decidability of typing can still be recovered, on expressive sub-fragments of the

	`Hyrql`	`FOQ`[30]	`Qunity`[51]	`DLPZ`[22]	`Spm`[44]	`QuGCL`[53]	λ-$\mathcal{S}_1$[25]
Hybrid	✓	✓	✓	✓	✗	✗	✗
General recursion	✓	✓	✗	✗	✓	✓	✗
Resource-aware	✓	✓	✗	✗	✗	✗	✗

Table 1. Comparison table between programming languages with quantum control

language, e.g., when `Hyrql` is restricted to finite types and terminating programs (Proposition 1).
- A compilation procedure (Algorithm 2) translating terms to (simply-typed) term-rewrite system (STTRS, [52]). This translation is correct (Propositions 3 and 5) and, crucially, the runtime-complexity of evaluating a `Hyrql`-program and the runtime-complexity of evaluating its STTRS-translation are linked (Theorem 4). As a consequence, most known techniques (e.g., interpretations [39], recursive path orderings [23,24], dependency pairs [4], size-change principle [37]) for analyzing the runtime of standard TRS can be used to show termination or infer bounds on the complexity of `Hyrql`-programs.

We illustrate the expressive power of the language and its resource analysis through several examples: implementation of basic quantum gates (Hadamard gate, Example 1), quantum control (quantum switch, Example 2), hybrid control flow (protocol *BB84* [11], Example 4), higher-order (map function, Example 12), and general recursion (Ackermann function, Example 10).

Related works. In Table 1, we provide a non-exhaustive comparison between the main families of programming languages with quantum control. Towards that end, we consider the three following criteria.

- *Hybrid* indicates whether the language treats classical data as first-class citizen and distinguishes between the languages `Hyrql`, `FOQ` [30], `Qunity` [51], and `DLPZ` [22], that can be used to analyze variational algorithms, and those which cannot. We have already discussed about `Spm` [44] and emphasized on its lack of hybrid design. The language `QuGCL` [53] tries to solve the problem of defining a quantum recursion on a purely-quantum language, which is known to be a hard problem [13]. Finally, λ-$\mathcal{S}_1$ [25] is an algebraic purely-quantum language aimed at guaranteeing unitarity.
- *General recursion* specifies whether the language includes unbounded recursion. The ability to handle this type of recursion is necessary for a resource analysis to be relevant (i.e., non trivial as in the case of strong normalizing languages). The language `Qunity` [51] is hybrid and features a compilation algorithm to quantum circuits. However it is terminating and hence does not support general recursion. `DLPZ` [22] is another hybrid language, inspired from `Spm`, with an adequate denotational semantics. However, `DLPZ` supports neither general inductive types (with the exception of natural numbers), nor general recursion. Note that this categorization does not distinguish between

$$\begin{array}{rrcl}
\text{(Values)} & v & ::= & x \mid |0\rangle \mid |1\rangle \mid c(\overrightarrow{v}) \mid \lambda x.t \mid \texttt{letrec}\, f\, x = t \mid \texttt{unit}(t) \mid \sum_{i=1}^{n} \alpha_i \cdot v_i \\
\text{(Terms)} & t & ::= & x \mid |0\rangle \mid |1\rangle \mid \texttt{qcase}\, t\, \{|0\rangle \to t_0\,, |1\rangle \to t_1\} \\
 & & & \mid c(\overrightarrow{t}) \mid \texttt{match}\, t\, \{c_1(\overrightarrow{x_1}) \to t_1\,, \ldots, c_n(\overrightarrow{x_n}) \to t_n\} \\
 & & & \mid \lambda x.t \mid \texttt{letrec}\, f\, x = t \mid \texttt{unit}(t) \mid t_1 t_2 \\
 & & & \mid \sum_{i=1}^{n} \alpha_i \cdot t_i \mid \texttt{shape}(t)
\end{array}$$

Table 2. Syntax of the `Hyrql` language

languages that have classical recursion and those that have quantum recursion, such as `QuGCL` [53].

– Finally, *Resource-aware* highlights if the language is designed towards resource analysis. This is the case of `Hyrql` as well as `FOQ` [30,31], which characterizes, under restrictions, quantum polynomial time as well as quantum polylogarithmic time [28]. However `Hyrql` has a greater expressive power than `FOQ` as it is not restricted to unitary operators. Moreover, all standard classical datatypes can be expressed in `Hyrql` whereas `FOQ` is restricted to lists of qubits.

Other studies have already been carried out on resource analysis in quantum programming languages. They use techniques (e.g., type systems) developed in the field of Implicit Computational Complexity (ICC, see [42]) to characterize quantum complexity classes [21], to infer the expected cost or expected value of a quantum program [6], or to infer upper bounds on quantum resources (depth and size of circuits) [18]. However these studies are restricted to classical control. For example, [21] studies the use of soft linear logic [35] on a quantum lambda calculus [47], [6] adapts expectation transformers on a language based on `QPL` [46], and [18] develops a dependent type system on a variant of the Quipper circuit description language [29]. Still in the field of ICC, our translation to STTRS follows a standard analysis pattern first introduced in [8], which consists in compiling higher-order programs to STTRS, that are more amenable for static/automatic complexity analysis. Hence our approach provides a first application of this methodology in the quantum setting.

2 Hybrid Quantum Language with General Recursion

We introduce `Hyrql` syntax, operational semantics, and type system along with illustrating examples.

2.1 Syntax

The syntax of Terms and Values in `Hyrql` is provided in Table 2. Terms feature variables, noted $x, y, f, \ldots$ taken from an infinite countable set of variables `Var`.

We denote by $\overrightarrow{e}$ a (finite and possible empty) sequence of elements $e_1, \dots, e_n$. Moreover, the notation $F(\overrightarrow{e})$ will be syntactic sugar for $F(e_1), \dots, F(e_n)$. Qubits are introduced through basis states $|0\rangle, |1\rangle$, and can be used in a quantum conditional $\texttt{qcase}\, t \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\}$, which corresponds to the superposition of t_0 and t_1 controlled by qubit t. Hyrql also features classical constructor symbols c, which can be used in constructor application $c(\overrightarrow{t})$ and through pattern-matching, sometimes abbreviated as $\texttt{match}_{1\le i\le n}\, t\, \{c_i(\overrightarrow{x_i}) \to t_i\}$. Each constructor symbol c comes with a fixed arity and is always fully applied. We assume the existence of standard inductive constructors for unit $()$, bits 0 and 1, natural numbers 0 and S, and lists $[\,]$ and $::$. For convenience, constructors are sometimes used in a infix notation. For example, tensor products will be written as $x \otimes y$ or list constructors will be written as $h :: t$.

Higher-order is featured via three constructs: a standard λ-abstraction $\lambda x.t$, a $\texttt{letrec}\, f\, x = t$ construct for general recursion, and a $\texttt{unit}(t)$ construct for unitary operators. Term application is denoted as $t_1 t_2$.

Given amplitudes $\alpha_i \in \mathbb{C}$, the term $\sum_{i=1}^{n} \alpha_i \cdot t_i$ represents a *superposition* of terms t_i. In the special case where $n = 2$, we just write $\alpha_1 \cdot t_1 + \alpha_2 \cdot t_2$. In a dual manner, the shape $\texttt{shape}(t)$ construct, introduced in [29], returns the classic structural information on data containing quantum information. For example, the shape of a quantum list is a classical data computing the list structure.

We denote $\mathrm{FV}(t)$ as the free variables of t. In order to avoid conflicts between free and bound variables we will always work up to α-conversion and use Barendregt's convention [9, p. 26] which consists in keeping all bound and free variable names distinct, even when this remains implicit.

Terms and Values have to be considered with respect to a vector space structure. Towards that end, we define an equivalence relation $\equiv$ in Table 3, relying on a notion of *equivalence contexts*, noted $C_\equiv$, terms with a hole $\diamond$ which are defined by the following grammar:

$$\begin{aligned} C_\equiv ::= \diamond \mid \texttt{qcase}\, C_\equiv \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\} \mid c(\overrightarrow{t_1}, C_\equiv, \overrightarrow{t_2}) \\ \mid \texttt{match}_{1\le i\le n}\, C_\equiv\, \{c_i(\overrightarrow{x_i}) \to t_i\} \mid tC_\equiv \mid C_\equiv t \mid t + C_\equiv \mid \texttt{shape}(C_\equiv) \end{aligned}$$

Let $C_\equiv[t]$ be the term obtained by filling the hole with t in $C_\equiv$.

The rules of Table 3 make the definition of the Σ construct unambiguous/sound, i.e., it corresponds exactly to repeated applications of $+$.

Example 1 (Quantum State and Hadamard Gate). We can define the orthogonal basis states as $|\pm\rangle \triangleq \frac{1}{\sqrt{2}} \cdot |0\rangle \pm \frac{1}{\sqrt{2}} \cdot |1\rangle$. The Hadamard gate can be encoded in our language as the unitary term:

$$\texttt{Had} \triangleq \texttt{unit}(\lambda x.\texttt{qcase}\, x \left\{|0\rangle \to |+\rangle\ , |1\rangle \to |-\rangle\right\})$$

Example 2 (Quantum Switch). The Quantum Switch [17] can be defined as

$$\texttt{QS} \triangleq \lambda f.\lambda g.\lambda x.\texttt{match}\, x \left\{c \otimes t \to \texttt{qcase}\, c \left\{|0\rangle \to |0\rangle \otimes f(g\, t)\, , |1\rangle \to |1\rangle \otimes g(f\, t)\right\}\right\}$$

$$t_1 + t_2 \equiv t_2 + t_1 \qquad t_1 + (t_2 + t_3) \equiv (t_1 + t_2) + t_3 \qquad 1 \cdot t \equiv t \qquad t + 0 \cdot t' \equiv t$$

$$\alpha \cdot (\beta \cdot t) \equiv \alpha\beta \cdot t \qquad \alpha \cdot (t_1 + t_2) \equiv \alpha \cdot t_1 + \alpha \cdot t_2 \qquad \alpha \cdot t + \beta \cdot t \equiv (\alpha + \beta) \cdot t$$

$$\texttt{qcase}\,(\sum_{j=1}^{m} \alpha_j \cdot s_j)\,\left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\} \equiv \sum_{j=1}^{m} \alpha_j \cdot (\texttt{qcase}\, s_j\, \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\})$$

$$c(\overrightarrow{t_1}, \sum_{j=1}^{n} \alpha_j \cdot s_j, \overrightarrow{t_2}) \equiv \sum_{j=1}^{m} \alpha_j \cdot c(\overrightarrow{t_1}, s_j, \overrightarrow{t_2})$$

$$\texttt{match}_{1\le i\le n}\,(\sum_{j=1}^{n} \alpha_j \cdot s_j)\,\{c_i(\overrightarrow{x_i}) \to t_i\} \equiv \sum_{j=1}^{n} \alpha_i \cdot (\texttt{match}_{1\le i\le n}\, s_j\, \{c_i(\overrightarrow{x_i}) \to t_i\})$$

$$C_{\equiv}[t] \equiv C_{\equiv}[t'] \qquad \text{when } t \equiv t'$$

Table 3. Equivalence relation $\equiv \subseteq$ Terms $\times$ Terms

Example 3 (Length of a list). Our language also allows us to define recursive functions such as the length of a list:

$$\texttt{len} \triangleq \texttt{letrec}\, f\, x = \texttt{match}\, x\, \left\{[\,] \to 0, h :: t \to S\,(f\, t)\right\}$$

Example 4 (Hybrid function — BB84 *[11]).* Hyrql is also able to use quantum and classical data simultaneously. For example, the well-known protocol *BB84* [11], creating a quantum key from a classical key can be implemented as follows:

$$\begin{aligned}
\texttt{not} &\triangleq \lambda q.\texttt{qcase}\, q\, \left\{|0\rangle \to |1\rangle\ , |1\rangle \to |0\rangle\right\}\\
\texttt{cc} &\triangleq \lambda b.\lambda f.\lambda q.\texttt{match}\, b\, \left\{0 \to q, 1 \to f\, q\right\}\\
\texttt{op} &\triangleq \lambda q.\lambda n.\texttt{match}\, n\, \left\{(x, h) \to \texttt{cc}\, h\, \texttt{Had}(\texttt{cc}\, x\, \texttt{not}\, q)\right\}\\
\texttt{keygen} &\triangleq \texttt{letrec}\, f\, l = \texttt{match}\, l\, \left\{[\,] \to [\,], h :: t \to (\texttt{op}\ |0\rangle\ h) :: (f\, t)\right\}
\end{aligned}$$

The term `cc` applies a gate f conditionally with classical control on a bit b. The term `op` reads a pair (x, h), and applies the NOT gate if $x = 1$, and then applies the Hadamard gate if $h = 1$. Finally, `keygen` generates the qubit list key.

2.2 Operational Semantics

In this section, we define the call-by-value operational semantics of Hyrql. Toward that end, we introduce the notions of pure terms and canonical form.

$$\frac{}{\texttt{qcase}\ |0\rangle \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\} \rightsquigarrow t_0}\ (\text{Qcase}_0) \qquad \frac{}{\texttt{qcase}\ |1\rangle \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\} \rightsquigarrow t_1}$$

$$\frac{}{\texttt{match}_{1\le i\le n}\, c_j(\overrightarrow{v})\,\{c_i(\overrightarrow{x_i}) \to t_i\} \rightsquigarrow t_j\{\overrightarrow{v}/\overrightarrow{x_j}\}}\ (\text{Match}) \qquad \frac{}{(\lambda x.t)v \rightsquigarrow t\{v/x\}}\ (\text{Lbd})$$

$$\frac{}{(\texttt{letrec}\, f\, x = t)v \rightsquigarrow t\{\texttt{letrec}\, f\, x = t/f, v/x\}}\ (\text{Fix}) \qquad \frac{}{\texttt{unit}(t)v \rightsquigarrow tv}\ (\text{Unit})$$

$$\frac{\sum_{i=1}^n \alpha_i \cdot t_i \in \texttt{CAN} \setminus \text{Value} \qquad t_i \rightsquigarrow^? t_i'}{\sum_{i=1}^n \alpha_i \cdot t_i \rightsquigarrow \sum_{i=1}^n \alpha_i \cdot t_i'}\ (\text{Can}) \qquad \frac{}{\texttt{shape}(|0\rangle) \rightsquigarrow ()}\ (\text{Shape}_0)$$

$$\frac{}{\texttt{shape}(|1\rangle) \rightsquigarrow ()}\ (\text{Shape}_1) \qquad \frac{}{\texttt{shape}(c(\overrightarrow{v})) \rightsquigarrow \tilde{c}(\texttt{shape}(\overrightarrow{v}))}\ (\text{Shape}_c)$$

$$\frac{\sum_{i=1}^n \alpha_i \cdot v_i \in \texttt{CAN}}{\texttt{shape}(\sum_{i=1}^n \alpha_i \cdot v_i) \rightsquigarrow \texttt{shape}(v_1)}\ (\text{Shape}_s) \qquad \frac{t \equiv t_1 \qquad t_1 \rightsquigarrow t_1' \qquad t_1' \equiv t'}{t \rightsquigarrow t'}\ (\text{Equiv})$$

$$\frac{t \rightsquigarrow t' \qquad E[t] \in \text{Pure terms}}{E[t] \rightsquigarrow E[t']}\ (E)$$

Table 4. Reduction rules of the language

Pure terms are defined by the following syntax:

$$\begin{aligned}(\text{Pure terms}) \quad p ::= {} & x \mid |0\rangle \mid |1\rangle \mid \texttt{qcase}\, p \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\} \mid c(\overrightarrow{p}) \\ & \mid \texttt{match}_{1\le i\le n}\, p\,\{c_i(\overrightarrow{x_i}) \to t_i\} \mid \lambda x.t \mid \texttt{letrec}\, f\, x = t \\ & \mid \texttt{unit}(t) \mid t_1 t_2 \mid \texttt{shape}(t)\end{aligned}$$

A *pure value* is a pure term that is a value.

Definition 1 (Canonical form). *The* canonical form *of a term t is any term t' such that $t' \equiv t$ and $t' = \sum_{i=1}^n \alpha_i \cdot p_i$, where $\alpha_i \neq 0$ and $p_i \equiv p_j$ implies $i = j$. Let $\texttt{CAN}$ be the set of canonical forms.*

As we will see shortly, canonical forms will ensure that no term with a null amplitude is ever reduced, i.e., reductions are meaningful. Furthermore, any pure value for either qubits or constructor terms will contain no superposition, giving the intuition of a *canonical basis*; this will be used to define the inner product in Subsection 2.3. Note that the naming *pure terms* comes from [25] and is not related to *pure states* in quantum computing as pure terms could semantically yield a mixed quantum state.

The notion of *evaluation contexts* will be used to define the operational semantics and is described by the following grammar:

$$\text{(Eval. contexts)} \quad E ::= \diamond \mid \texttt{qcase}\, E \left\{ |0\rangle \to t_0 \,, |1\rangle \to t_1 \right\} \mid c(\overrightarrow{t}, E, \overrightarrow{v}) \\ \mid \texttt{match}_{1\le i\le n}\, E \,\{c_i(\overrightarrow{x_i}) \to t_i\} \mid tE \mid Ev \mid \texttt{shape}(E)$$

Again, let $E[t]$ be the term obtained by filling the hole $\diamond$ with t in E.

The operational semantics of `Hyrql` is described in Table 4 as a reduction relation $\rightsquigarrow \subseteq$ Terms $\times$ Terms, where $\{t/x\}$ denotes the standard substitution of variable x by term t. The reduction implements a *call-by-value* strategy.

In all rules of Table 4, except for the (Can) and (Equiv) rules, the left-hand-side term is a pure term (i.e., not a superposition). In the rule (Can), $t \rightsquigarrow^{?} t'$ holds if either $t \rightsquigarrow t'$ or, $\neg(\exists t'',\ t \rightsquigarrow t'')$ and $t' = t$. Intuitively, it means that we apply one reduction in each element of a superposition, when possible. Moreover, any summation must be expressed in a canonical form before reducing, which avoids reducing a subterm with a null amplitude, or more generally to reduce two identical terms that would sum up to 0. The goal is to avoid reductions that have no physical meaning, as this would invalidate all resource analysis results.

We define $\rightsquigarrow^*$ as the reflexive and transitive closure of $\rightsquigarrow$ modulo the equivalence $\equiv$. For $k \in \mathbb{N}$, we also write $t \rightsquigarrow^{\le k} t'$, when t reduces to t' in at most k steps. Finally, a term t *terminates*, if any chain of reduction starting from t reaches a value, meaning that $t \rightsquigarrow^{\le k} v$ holds for some k.

Example 5. Let us consider the `Had` program from Example 1, we can check that it gives the desired result when applying it to $|0\rangle$:

$$\begin{aligned} \texttt{Had}\,|0\rangle &\rightsquigarrow (\lambda x.\texttt{qcase}\, x \left\{ |0\rangle \to |+\rangle \,, |1\rangle \to |-\rangle \right\})\ |0\rangle && \text{via (Unit)} \\ &\rightsquigarrow \texttt{qcase}\ |0\rangle \left\{ |0\rangle \to |+\rangle \,, |1\rangle \to |-\rangle \right\} && \text{via (Lbd)} \\ &\rightsquigarrow |+\rangle && \text{via (Qcase}_0) \end{aligned}$$

Since the Hadamard gate is its own inverse we can recover $|0\rangle$. First, let the term $h_v = \texttt{qcase}\, v \left\{ |0\rangle \to |+\rangle \,, |1\rangle \to |-\rangle \right\}$, then:

$$\begin{aligned} \texttt{Had}\,|+\rangle &\rightsquigarrow (\lambda x.\texttt{qcase}\, x \left\{ |0\rangle \to |+\rangle \,, |1\rangle \to |-\rangle \right\})\ |+\rangle \\ &\rightsquigarrow h_{|+\rangle} \equiv \frac{1}{\sqrt{2}} \cdot h_0 + \frac{1}{\sqrt{2}} \cdot h_1 \\ &\rightsquigarrow \frac{1}{\sqrt{2}} \cdot |+\rangle + \frac{1}{\sqrt{2}} \cdot |-\rangle \equiv |0\rangle \end{aligned}$$

Example 6. Given a qubit list $l = |0\rangle :: |1\rangle :: |+\rangle :: [\ ]$, one can verify that $\texttt{shape}(l) \rightsquigarrow^* () :: () :: () :: [\ ]$. The result is a list of same length as l, with no quantum data anymore, hence it can be treated non-linearly. Note that such operation is independent from the value of each qubit in the list.

2.3 Type System

Types in `Hyrql` are provided by the following grammar.

$$\text{(Types)} \qquad T ::= \text{Qbit} \mid B \mid T \multimap T \mid T \Rightarrow T \mid T \leftrightarrow T$$

Qbit is the type for qubits and B is a *constructor type* from a fixed set $\mathbb{B}$. A *basic type*, noted $B^{|\rangle}$, is either a qubit type, or a constructor type B, i.e., $B^{|\rangle} \in \{\text{Qbit}\} \cup \mathbb{B}$. Each constructor type $B \in \mathbb{B}$ comes with a fixed set of constructor symbols $\text{Cons}(B)$ such that $B \neq B'$ implies $\text{Cons}(B) \cap \text{Cons}(B)' = \emptyset$. Each constructor symbol $c \ \in \text{Cons}(B)$ of arity n has a fixed signature $c :: B_1^{|\rangle}, \ldots, B_n^{|\rangle} \to B$. We just write $c :: B$ when $n = 0$. In what follows, we will consider the following constructor types: a unit type $\mathbb{1}$ with a unique constructor $() :: \mathbb{1}$; tensor types $B_1^{|\rangle} \otimes B_2^{|\rangle}$ for given types $B_1^{|\rangle}, B_2^{|\rangle}$ with a unique constructor $\texttt{tens} :: B_1^{|\rangle}, B_2^{|\rangle} \to B_1^{|\rangle} \otimes B_2^{|\rangle}$, denoted $x \otimes y$ when applied; natural numbers $\texttt{nat}$ with constructors $0 :: \texttt{nat}$ and $S :: \texttt{nat} \to \texttt{nat}$; lists $\texttt{list}(B^{|\rangle})$ for a given type $B^{|\rangle}$, with constructors $[\ ] :: \texttt{list}(B^{|\rangle})$ and $\texttt{cons} :: B^{|\rangle}, \texttt{list}(B^{|\rangle}) \to \texttt{list}(B^{|\rangle})$, denoted $h :: t$ when applied. We consider that $\mathbb{B}$ always contains the unit type $\mathbb{1}$.

The language also features three distinct *higher-order* types for linear, non-linear, and unitary functions, denoted respectively by $\multimap$, $\Rightarrow$, and $\leftrightarrow$.

In the typing discipline, it will be useful to distinguish between types based on whether or not they contain quantum data. The latter must follow the laws of quantum mechanics (no-cloning), whereas classical types are more permissive. Toward that end, we define the set of *quantum constructor types* $\mathbb{B}_Q$ by $\mathbb{B}_Q \triangleq \{B \in \mathbb{B} \mid \exists c :: B_1^{|\rangle}, \ldots, B_n^{|\rangle} \to B, \exists 1 \leq i \leq n, B_i^{|\rangle} \in \mathbb{B}_Q \cup \{\text{Qbit}\}\}$. A quantum constructor type $B_Q \in \mathbb{B}_Q$ has (at least) a constructor symbol with a type Qbit or (inductively) a quantum constructor type in its signature. The set $\mathbb{B}_C$ of *classical constructor types* B_C is defined by $\mathbb{B}_C \triangleq \mathbb{B} \setminus \mathbb{B}_Q$. For example, $\mathbb{1}, \texttt{nat} \in \mathbb{B}_C$ whereas $\texttt{list}(\text{Qbit}) \in \mathbb{B}_Q$.

Now, we can split the types between quantum types Q and classical types C.

$$Q ::= \text{Qbit} \mid B_Q \qquad C ::= B_C \mid T \multimap T \mid T \Rightarrow T \mid T \leftrightarrow T$$

We define typing contexts as follows:

$$\Gamma, \Delta \quad ::= \quad \varnothing \mid \{x : T\} \mid \{[x : T]\} \mid \Gamma \cup \Delta,$$

where $[x : T]$ is called a *boxed variable*, indicating that x is a linear variable captured by $\texttt{shape}$. As we have mentioned, the language features both classical (duplicable and erasable) and quantum (non duplicable and non erasable) data. A typing judgment is thus written $\Gamma; \Delta \vdash t : T$, describing the fact that under *non-linear context* Γ and *linear context* Δ, the term t has type T. Such judgment is then derived inductively following the rules of Table 5. Whenever we write Γ, Δ or $\Gamma; \Delta$, it is assumed that Γ and Δ are compatible, that is, no variable appears twice in $\Gamma \cup \Delta$, considering that a boxed variable is distinct from the unboxed one. We use the shorthand notation $\overrightarrow{x} : \overrightarrow{T}$ for $x_1 : T_1, \ldots, x_n : T_n$. The key points of the type system are the following:

- In both (qcase) and (sup), we ask that terms are orthogonal, written $s \perp t$ and formalized in Definition 2. This is akin to the orthogonality condition that we found in quantum physics, and similar to other existing languages [25,38,22].

$$\frac{}{\varGamma; x : T \vdash x : T} \qquad \frac{}{\varGamma, x : C; \varnothing \vdash x : C} \qquad \frac{}{\varGamma; \varnothing \vdash |0\rangle : \text{Qbit}} \qquad \frac{}{\varGamma; \varnothing \vdash |1\rangle : \text{Qbit}}$$

$$\frac{\varGamma; \varDelta \vdash t : \text{Qbit} \qquad \varGamma; \varDelta' \vdash t_0 : T \qquad \varGamma; \varDelta' \vdash t_1 : T \qquad t_0 \perp t_1}{\varGamma; \varDelta, \varDelta' \vdash \texttt{qcase}\, t \left\{ |0\rangle \to t_0 \,, |1\rangle \to t_1 \right\} : T} \text{ (qcase)}$$

$$\frac{\varGamma; \varDelta_i \vdash t_i : B_i^{|\rangle} \qquad c :: B_1^{|\rangle}, \dots, B_n^{|\rangle} \to B}{\varGamma; \varDelta_1, \dots, \varDelta_n \vdash c(t_1, \dots, t_n) : B}$$

$$\frac{\begin{array}{cc} c_i :: \overrightarrow{C_i}, \overrightarrow{Q_i} \to B & \varGamma; \varDelta \vdash t : B \\ \text{Cons}(B) = \{c_i\}_{i=1}^n & \varGamma, \overrightarrow{y_i} : \overrightarrow{C_i}; \varDelta', \overrightarrow{z_i} : \overrightarrow{Q_i} \vdash t_i : B^{|\rangle} \end{array}}{\varGamma; \varDelta, \varDelta' \vdash \texttt{match}\, t \left\{ c_1(\overrightarrow{y_1}, \overrightarrow{z_1}) \to t_1 \,, \dots, c_n(\overrightarrow{y_n}, \overrightarrow{z_n}) \to t_n \right\} : B^{|\rangle}} \text{ (match)}$$

$$\frac{\varGamma; \varDelta, x : T \vdash t : T'}{\varGamma; \varDelta \vdash \lambda x.t : T \multimap T'} \qquad \frac{\varGamma, x : C; \varDelta \vdash t : T}{\varGamma; \varDelta \vdash \lambda x.t : C \Rightarrow T} \qquad \frac{\varGamma, f : T; \varnothing \vdash \lambda x.t : T}{\varGamma; \varnothing \vdash \texttt{letrec}\, f\, x = t : T}$$

$$\frac{\varGamma; \varnothing \vdash t : Q \multimap Q' \qquad t \text{ is unitary}}{\varGamma; \varnothing \vdash \texttt{unit}(t) : Q \leftrightarrow Q'} \qquad \frac{\varGamma; \varDelta \vdash t_1 : T \multimap T' \qquad \varGamma; \varDelta' \vdash t_2 : T}{\varGamma; \varDelta, \varDelta' \vdash t_1 t_2 : T'}$$

$$\frac{\varGamma; \varDelta \vdash t_1 : C \Rightarrow T \qquad \varGamma; \varnothing \vdash t_2 : C}{\varGamma; \varDelta \vdash t_1 t_2 : T} \qquad \frac{\varGamma; \varnothing \vdash t_1 : Q_1 \leftrightarrow Q_2 \qquad \varGamma; \varDelta \vdash t_2 : Q_1}{\varGamma; \varDelta \vdash t_1 t_2 : Q_2}$$

$$\frac{\varGamma; \varDelta \vdash t_i : Q \qquad \sum_{i=1}^n |\alpha_i|^2 = 1 \qquad \forall i \neq j,\ t_i \perp t_j}{\varGamma; \varDelta \vdash \sum_{i=1}^n \alpha_i \cdot t_i : Q} \text{ (sup)}$$

$$\frac{\varGamma; \varDelta \vdash t : B^{|\rangle}}{\varGamma, [\varDelta]; \varnothing \vdash \texttt{shape}(t) : \texttt{shape}(B^{|\rangle})} \text{ (shape)}$$

$$\frac{\varGamma, [x : B^{|\rangle}]; \varDelta, y : B^{|\rangle} \vdash t : T \qquad \sigma = \{y/x\}}{\varGamma; \varDelta, y : B^{|\rangle} \vdash \sigma(t) : T} \text{ (contr)} \qquad \frac{\varGamma; \varDelta \vdash t : T \qquad t \equiv t'}{\varGamma; \varDelta \vdash t' : T} \text{ (equiv)}$$

Table 5. Typing rules of the language

– In the rule (match), we assume that the inputs of each constructor are ordered, containing first *classical*, then *quantum* variables, enabling a hybrid behaviour.
– The rule (shape) extracts the classical structure of a quantum term, boxes any variable in Δ, and puts it in Γ. The term obtained has an empty linear context and a type in $\mathbb{B}_C$ as discussed in Definition 4, and thus can be seen and used classically.
– The rule (contr) binds a boxed variable x with a linear variable of same type y, allowing to read both the quantum data and the classic structural information of y.

In order for the rules to be well-defined, we need to introduce two predicates: one about the *orthogonality* of terms, which allows us to put them into superposition, and one about the *unitarity* of terms, meaning that a term describes a unitary operation. Note that the rules requiring a predicate apply it only to well-typed terms, thus our type system is not ill-defined.

To do so, we first introduce *inner products*. It is defined for well-typed, terminating and closed terms as follows:

$$\langle s, t\rangle = \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_i \beta_j^* \delta_{v_i, w_j},$$

where $s \rightsquigarrow^* \sum_{i=1}^{n} \alpha_i \cdot v_i \in \texttt{CAN}$ $t \rightsquigarrow^* \sum_{j=1}^{m} \beta_j \cdot w_j \in \texttt{CAN}$, and $\delta_{v,w}$ is the Kronecker symbol. While the inner product requires a reduction to a canonical form, we will see later than this can be generalized to any term. Given a context Δ, a Δ-context substitution is a substitution σ such that for any $x : T$ with $x : T \in \Delta$ or $[x : T] \in \Delta$, $\{v/x\} \in \sigma$, where v is a well-typed closed value of type T'. This definition ensures that the substitution is well-founded, to preserve typing of $\sigma(t)$ and yield a closed term.

Definition 2 (Orthogonality). *Let $\Gamma; \Delta \vdash s : B^{|\rangle}$ and $\Gamma; \Delta \vdash t : B^{|\rangle}$ be two well-typed terms. We say that s and t are orthogonal, written $s \perp t$, if for any $\Gamma \cup \Delta$-context substitution σ:*
- *$\sigma(\texttt{shape}(s)) \rightsquigarrow^* v$, $\sigma(\texttt{shape}(t)) \rightsquigarrow^* w$ and $v = w$;*
- *$\langle \sigma(s), \sigma(t)\rangle = 0$.*

While the second condition computes the inner product to check orthogonality, the first one ensures both terms have the same classical structure, and only differ on their quantum part; for example, orthogonality between lists requires both lists to be of the same size.

This inner product allows us to define *isometries* as terms preserving the inner product. A well-typed term $\Gamma; \Delta \vdash t : B_1^{|\rangle} \rightarrowtail B_2^{|\rangle}$ is said to be an isometry if for any well-typed closed value v, w of type $B_1^{|\rangle}$, and any $\Gamma \cup \Delta$-context substitution, $\langle \sigma(t)v, \sigma(t)w\rangle = \langle v, w\rangle$. In particular, this requires that t terminates over any substitution and any input. In the same fashion, t is said to be *surjective*, if for any $\Gamma \cup \Delta$-context substitution and any $\varnothing; \varnothing \vdash w : B_2^{|\rangle}$, there exists

$\varnothing; \varnothing \vdash v : B_1^{|\rangle}$ such that $\sigma(t)v \rightsquigarrow^* w$. Altogether, this allows us to define unitary terms.

Definition 3 (Unitary term). *Let $\Gamma; \Delta \vdash t : Q \multimap Q'$. We say that t is unitary, if it is surjective and an isometry.*

Finally, to properly define the rule (shape), we introduce two notations. First, given a context $\Delta = x_1 : A_1, \dots, x_n : A_n$, we use the notation $[\Delta]$ for $[x_1 : A_1], \dots, [x_n : A_n]$; we also define the shape of a basic type below.

Definition 4. *We define the shape of a basic type as follows:*

$$\texttt{shape}(\mathrm{Qbit}) \triangleq \mathbb{1} \qquad \texttt{shape}(B) \triangleq \tilde{B}$$

where $\tilde{B} \in \mathbb{B}$ *is such that* $\mathrm{Cons}(\tilde{B}) \triangleq \{\tilde{c} :: \texttt{shape}(B_1^{|\rangle}, \dots, B_n^{|\rangle}) \to \tilde{B} \mid c :: B_1^{|\rangle}, \dots, B_n^{|\rangle} \to B \in \mathrm{Cons}(B)\}$.

By construction, it always holds that $\texttt{shape}(B^{|\rangle}) \in \mathbb{B}_C$. Therefore, any variable with such type is classical, and can be used non-linearly. The shape of a term is the classical structure of said term. For instance, the shape of a list of qubits is a list of units, i.e., $\texttt{shape}(\texttt{list}(\mathrm{Qbit})) = \texttt{list}(\mathbb{1})$.

Example 7. Coming back on Example 1, one can type $\varnothing; \varnothing \vdash \texttt{Had} : \mathrm{Qbit} \leftrightarrow \mathrm{Qbit}$. Indeed, $|+\rangle \perp |-\rangle$, thus `qcase` is typable, and the λ-abstraction satisfies Definition 3.

Example 8. The term `len` from Example 3 cannot be typed if x has a quantum type, as t is used non-linearly. However, for any $B \in \mathbb{B}_C$, `len` is a well-typed closed term of type $\texttt{list}(B) \Rightarrow \texttt{nat}$. Given a list of qubits, we can still measure its length: $t = \lambda y.(\texttt{len}(\texttt{shape}(y)) \otimes y)$ is a well-typed closed term of type $\texttt{list}(\mathrm{Qbit}) \multimap \texttt{nat} \otimes \texttt{list}(\mathrm{Qbit})$.

The typing derivations of the examples seen thus far are available in Appendix A.

3 Properties of the language

This section is devoted to checking that `Hyrql` enjoys useful properties such as confluence, to studying the complexity of the orthogonality predicate, and to exploring the behaviour of linear and unitary functional terms.

First, by design of the type system, the non-linear context Γ can be expanded and typing still holds.

Lemma 1 (Weakening). *Let $\Gamma; \Delta \vdash t : T$ be a well-typed term. Then $\Gamma, \Gamma'; \Delta \vdash t : T$ holds for any context Γ' compatible with Γ, Δ.*

The reduction relation $\rightsquigarrow$ is confluent up to equivalence for well-typed terms.

Theorem 1 (Confluence). *Given a well-typed term t if there exist t_1 and t_2 such that $t \rightsquigarrow^* t_1$ and $t \rightsquigarrow^* t_2$, then there exist t_3 and t_4 such that $t_1 \rightsquigarrow^* t_3$, $t_2 \rightsquigarrow^* t_4$ and $t_3 \equiv t_4$.*

Equivalence is needed because of the reduction rule (Equiv). Typing is also needed for (Shape$_s$), to guarantee that we only have superpositions with terms of the same shape. In particular, this implies that any terminating term has a unique normal form, up to equivalence. This will be useful for progress (Lemma 3).

Lemma 2 (Canonical form for typed terms). *Let $\Gamma; \Delta \vdash t : T$ be a well-typed term. Then t has a canonical form $\sum_{i=1}^{n} \alpha_i \cdot t_i$, and this canonical form is unique up to reordering and equivalence on the t_i. Furthermore, $\Gamma; \Delta \vdash t_i : T$.*

The unicity of canonical forms is useful to ensure orthogonality is not ill-defined. However, orthogonality is undecidable in the general case. Indeed, it is as hard as the Universal Halt Problem [26]. The reader may look at [41] for an introduction on the arithmetical hierarchy. Note that, to be able to compute the inner product, we have to compute addition, multiplication and nullity check for complex numbers. Recall that algebraic numbers are complex numbers in $\mathbb{C}$ that are roots of a polynomial in $\mathbb{Q}[X]$, and write their sets as $\bar{\mathbb{C}}$. In the field of algebraic numbers, equality is decidable and product and sum are computable in polynomial time [32]. We thus restrict ourselves to terms that contain only amplitudes $\alpha \in \bar{\mathbb{C}}$.

Theorem 2 (Undecidability of orthogonality). *Deciding orthogonality between two well-typed terms is Π_2^0-complete.*

This is not surprising as our language allows for general recursion. However, it can be easily restricted to decidable cases. In order to do so, we introduce for basic types a *type depth* $d(B^{|\rangle}) \in \mathbb{N} \cup \{+\infty\}$, defined as follows:

$$d(\mathrm{Qbit}) = 1 \qquad d(B) = \max_{\substack{c :: B_1^{|\rangle}, \dots, B_n^{|\rangle} \to B \\ c \in \mathrm{Cons}(B)}} \left(\sum_{i=1}^{n} d(B_i^{|\rangle})\right) + 1$$

Note that $d(B^{|\rangle})$ is infinite for inductive types. We call *finite types* any type with a finite depth.

Proposition 1. *Given terms s, t that are well-typed, of finite type, closed, and terminating, it is decidable whether they are orthogonal. More, if s, t terminate in polynomial time, then orthogonality can be computed in polynomial time.*

In previous works on `Spm` [15,14] the syntax of the language was restricted enough to ensure the decidability of orthogonality. As `Spm` is a strict sublanguage of `Hyrql`, this undecidability result is still true for this fragment of our language.

Typing implies that the values are the normal forms of the language, up to equivalence.

Lemma 3 (Progress). *Let $\varnothing; \varnothing \vdash t : T$ be a closed term, either t is equivalent to a value, or t reduces.*

Typing is also preserved by reduction, provided we consider pure or terminating terms.

Lemma 4 (Subject reduction). *Let $\Gamma; \Delta \vdash t : T$ be a well-typed term, and $t \rightsquigarrow t'$. If t terminates or t is pure, then $\Gamma; \Delta \vdash t' : T$.*

Remark 1. `Hyrql` fails to derive suject reduction for a non-terminating and non-pure term. This is the case with the following example:

$$s_{a,b} = \texttt{qcase}\,((\texttt{letrec}\, f\, x = f x)\,|0\rangle)\,\{|0\rangle \to |0\rangle \otimes (a \otimes b)\,, |1\rangle \to |1\rangle \otimes (a \otimes b)\}$$
$$t = \texttt{match}\,\left(\frac{1}{\sqrt{2}} \cdot (|0\rangle \otimes |0\rangle) + \frac{1}{\sqrt{2}} \cdot (|1\rangle \otimes |1\rangle)\right)\{a \otimes b \to s_{a,b}\}$$

t is well typed, but reducing it through (Equiv) gives $t \rightsquigarrow \frac{1}{\sqrt{2}} \cdot s_{0,0} + \frac{1}{\sqrt{2}} \cdot s_{1,1}$, which is not typable: as $s_{0,0}$ and $s_{1,1}$ do not terminate, it is impossible to check their orthogonality, thus to type their superposition.

Finally, our type system also gives us properties on terms with a linear functional type. First, it behaves as a linear operator with respect to its inputs.

Proposition 2 (Linear functional terms). *Let $\varnothing; \varnothing \vdash t : T_1 \multimap T_2$ such that $t\, w$ terminates for any input $\varnothing; \varnothing \vdash w : T_1$. For any $\varnothing; \varnothing \vdash \sum_{i=1}^{n} \alpha_i \cdot v_i : T_1$, there exist t_1, t_2 such that $t\,(\sum_{i=1}^{n} \alpha_i \cdot v_i) \rightsquigarrow^* t_1$ and $\sum_{i=1}^{n} \alpha_i \cdot t\, v_i \rightsquigarrow^* t_2$, and $t_1 \equiv t_2$.*

Furthermore, quantum linear terms satisfy our isometry definition.

Proposition 3 (Quantum linear terms are isometries). *Let $\Gamma; \Delta \vdash t : Q_1 \multimap Q_2$; if it terminates for any input, then t is an isometry.*

This result allows us to simplify typing of unitary terms, by only checking surjectivity. While this is still undecidable, `Spm` [45] gives a decidable criterion implying surjectivity. Therefore, we could obtain a decidable fragment of unitarity for a subset of terms expressible in `Spm`.

4 Resource Analysis based on Simply-Typed Term Rewrite Systems

We are now interested in analyzing resource consumption in `Hyrql`, for example, showing the termination of a given term, bounding the complexity of reducing a term, or characterizing a given complexity class. Towards that end, we will introduce a semantics-preserving and complexity-preserving compilation algorithm `TranslateEntry` from terms of `Hyrql` to Simply-Typed Term Rewrite

$$\begin{array}{rrcl}
\text{(Types)} & \mathtt{A} & ::= & \mathtt{D} \mid \mathtt{A}_1 \times \cdots \times \mathtt{A}_n \to \mathtt{A} \\
\text{(Terms)} & \mathtt{t} & ::= & \mathtt{x}^{\mathtt{A}} \mid \mathtt{f}^{\mathtt{A}} \mid \mathtt{c}^{\mathtt{A}} \mid (\mathtt{t}^{\mathtt{A}_1 \times \cdots \times \mathtt{A}_n \to \mathtt{A}}({\mathtt{t}_1}^{\mathtt{A}_1}, \ldots, {\mathtt{t}_n}^{\mathtt{A}_n}))^{\mathtt{A}} \mid (\sum_{i=1}^{n} \alpha_i \cdot {\mathtt{t}_i}^{\mathtt{A}})^{\mathtt{A}} \\
\text{(Patterns)} & \mathtt{p} & ::= & \mathtt{x}^{\mathtt{A}} \mid (\mathtt{c}^{\mathtt{D}_1 \times \cdots \times \mathtt{D}_n \to \mathtt{D}}({\mathtt{p}_1}^{\mathtt{D}_1}, \ldots, {\mathtt{p}_n}^{\mathtt{D}_n}))^{\mathtt{D}} \\
\text{(Values)} & \mathtt{v} & ::= & \mathtt{v}^{\mathtt{A}} \mid \mathtt{f}^{\mathtt{A}} \mid \mathtt{c}^{\mathtt{A}} \mid (\mathtt{c}^{\mathtt{D}_1 \times \cdots \times \mathtt{D}_n \to \mathtt{D}}({\mathtt{v}_1}^{\mathtt{D}_1}, \ldots, {\mathtt{v}_n}^{\mathtt{D}_n}))^{\mathtt{D}} \mid (\sum_{i=1}^{n} \alpha_i \cdot {\mathtt{v}_i}^{\mathtt{A}})^{\mathtt{A}}
\end{array}$$

Table 6. Syntax of a STTRS

Systems (STTRS, for short) [52]. By complexity-preserving, we mean that each reduction step in Hyrql can be simulated by at most a constant number of reduction steps in the STTRS (Proposition 4). The significance of this result is that all known tools and techniques developed for analyzing the complexity/termination of Term Rewrite Systems can be reused to directly obtain complexity/termination guarantees on hybrid quantum programs. Such techniques have been deeply studied for term rewriting systems. Among the important references on termination, we can cite *polynomial interpretations* [36], *recursive path orderings* [23,24], *dependency pairs* [4], or *size-change termination* [37]; while on complexity, we can cite *quasi-interpretation* [12], *sup-interpretation* [39], *higher-order interpretations* [8], or *polynomial path orders* [40,5].

4.1 Syntax and Semantics of STTRS

The syntax of terms and types of STTRS in Table 6. Any type $\mathtt{A}$ is either a data type $\mathtt{D}$, which comes from a fixed and countable set, or a functional type, represented with at least one occurence of $\to$. $\mathtt{D}$ is assumed to contain the datatypes used in Hyrql, in particular, $\{\text{Qbit}\} \cup \mathbb{B} \subseteq \mathtt{D}$. The notation $\mathtt{t}^{\mathtt{A}}$ indicates that term $\mathtt{t}$ has type $\mathtt{A}$. The language features variables $\mathtt{x}^{\mathtt{A}}$, function symbols $\mathtt{f}^{\mathtt{A}} \in \mathcal{F}$, constructors $\mathtt{c}^{\mathtt{A}}$, applications $(\mathtt{t}^{\mathtt{A}_1 \times \cdots \times \mathtt{A}_n \to \mathtt{A}}({\mathtt{t}_1}^{\mathtt{A}_1}, \ldots, {\mathtt{t}_n}^{\mathtt{A}_n}))^{\mathtt{A}}$ of either functions or constructors, and superpositions $(\sum_{i=1}^{n} \alpha_i \cdot \mathtt{t}_i^{\mathtt{A}})^{\mathtt{A}}$. The set of function symbols $\mathcal{F}$ contains two fixed function symbols $\mathtt{unit}^{((Q \to Q') \times Q) \to Q'}$ and $\mathtt{shape}^{B^{|\rangle} \to B}$.

We can now define formally what a STTRS is, by defining rewrite rules. Note that we require rewrite rules to be orthogonal, meaning intuitively that a term cannot reduce with two distinct rules, and any variable appears once in the left part of a rule. For a more formal definition, the reader may look at [7, Ch.6].

Definition 5 (STTRS Definition). *We define $\to$ as a binary relation on the terms of a STTRS, where $\mathtt{l} \to \mathtt{r}$ implies:*

- $\mathtt{l}$ *and* $\mathtt{r}$ *are of the same type* $\mathtt{A}$*,*
- $\mathrm{FV}(\mathtt{l}) \supseteq \mathrm{FV}(\mathtt{r})$*,*
- $\mathtt{l}$ *is of the shape* $\mathtt{l} = \mathtt{f}(\overrightarrow{\mathtt{p}})$*, where* $\mathtt{f} \in \mathcal{F}$ *and the* $\mathtt{p}_i$ *are patterns.*

$\mathtt{l} \to \mathtt{r}$ is called a rewrite rule.

A simply-typed term rewriting system is a set $\mathcal{R}$ of orthogonal rewrite rules such that each symbol $\mathtt{f}$ has the same arity in every rule defining it.

The reduction of STTRS is directly adapted from [8]: we say that $\mathtt{s}$ rewrites to $\mathtt{t}$, written $\mathtt{s} \to_{\mathcal{R}} \mathtt{t}$, if there exists a rewrite rule $\mathtt{l} \to \mathtt{r} \in \mathcal{R}$, a substitution σ, and a TRS-context $\mathtt{C}$ such that $\mathtt{s} = \mathtt{C}[\sigma \mathtt{l}]$ and $\mathtt{t} = \mathtt{C}[\sigma \mathtt{r}]$. To take superposition into account, we need to define an equivalence relation $\equiv_{\mathcal{R}}$ on terms of a STTRS and we say that a superposition rewrites to another if at least one of its components can be rewritten and, similarly to rule (Can) of the reduction of `Hyrql`, the result of rewriting a superposition is the superposition of the rewritings of each component that can be rewritten: intuitively, $\sum_{i=1}^{n} \alpha_i \cdot \mathtt{s}_i \to_{\mathcal{R}} \sum_{i=1}^{n} \alpha_i \cdot \mathtt{t}_i$ where $\alpha_i \neq 0$ and $\mathtt{s}_i$ are pure terms, if for any $1 \leq i \leq n$, either $\mathtt{s}_i$ is a value and $\mathtt{s}_i = \mathtt{t}_i$, or $\mathtt{s}_i \to_{\mathcal{R}} \mathtt{t}_i$ according to the standard definition. The full formal definition of the semantics of STTRS is given in Definition 8 in Appendix.

4.2 Transpilation from Terms to STTRS

Now that we have introduced STTRS, our main goal is to provide a link between both worlds. We provide an algorithm that translates any well-typed term of `Hyrql` into a STTRS, inductively on the syntax, mainly by creating branches for each `match` and `qcase`. The algorithm `TranslateEntry` (Algorithm 3 in Appendix) satisfies the following result:

Proposition 4. *For any well-typed* `Hyrql` *term t, $\mathcal{R} = \mathtt{TranslateEntry}(t)$ is a well-defined STTRS. Furthermore, if $\mathcal{R}$ terminates on any input s in time $f(|s|)$, then $t\,s$ reduces in $\mathcal{O}(|s|^3 f(|s|))$ and $\Omega(f(|s|))$ steps.*

In particular, this means that termination of a STTRS produced by the algorithm `TranslateEntry` implies the termination of the `Hyrql` term; and if said STTRS terminates in polynomial time, then so does the `Hyrql` term. Therefore termination and complexity analysis techniques for TRS and STTRS directly give corresponding results for our language.

Note that `TranslateEntry` transforms terms into a subset of terms called *admissible terms* while preserving semantics. This transformation is responsible for the $|s|^3$ factor. `TranslateAdmissible` (Algorithm 2 in Appendix) is then called on an admissible term and produces a rewrite system whose runtime complexity is linear in that of the admissible term.

4.3 Illustrating Examples

In this section, we develop examples of `Hyrql` programs to show how they translate into STTRS and study their complexity, termination.

Example 9 (Hadamard function). We start by giving a feel of the translation of `Hyrql` into STTRS with the Hadamard program from Example 1. Doing so results in the following STTRS:

$$\mathcal{R} = \{\mathtt{had}_{\mathtt{R}}\, |0\rangle \to |+\rangle\,, \mathtt{had}_{\mathtt{R}}\, |1\rangle \to |-\rangle\},$$

where $\mathtt{had_R}$ is the function symbol corresponding to the initial term. This straightforwardly gives us the expected behavior.

Example 10 (Ackermann's function). As our language is hybrid, it can encode well-known classical recursive functions. For example, *Ackermann's function* can be written into `Hyrql` by the term t given below, with its corresponding STTRS $\mathcal{R}$:

$$t = \mathtt{letrec}\, f\, m = \lambda n.\mathtt{match}\, m \left\{ \begin{array}{c} 0 \to S\, n \\ S\, m' \to \mathtt{match}\, n \left\{ \begin{array}{c} 0 \to f\, m'\, 1 \\ S\, n' \to f\, m'\, (f\, (S\, m')\, n') \end{array} \right\} \end{array} \right\}$$

$$\mathcal{R} = \left\{ \begin{array}{c} \mathtt{ack}(0, n) \to S\, n \\ \mathtt{ack}(S\, m, 0) \to \mathtt{ack}(m, 1) \\ \mathtt{ack}(S\, m, S\, n) \to \mathtt{ack}(m, \mathtt{ack}(S\, m, n)) \end{array} \right\}$$

It is known that we can prove termination of such STTRS using *lexicographic path ordering* [24]. Using Proposition 4, we can conclude that t terminates over any input. Termination can also be proven using the *size-change principle* [37].

Example 11. Path orderings can also be combined with external techniques to obtain complexity bounds. For instance, the term `len` from Example 3 can be translated into the following STTRS:

$$\mathcal{R} = \{\mathtt{f}\, [\,] \to 0, \mathtt{f}\, (h :: t) \to S\, (\mathtt{f}\, t)\}$$

Termination of such STTRS can be proven easily using a path ordering. Furthermore, let us write the following assignment:

$$||0|| = 0 \quad ||S\, n|| = ||n|| + 1 \quad ||[\,]|| = 0 \quad ||h :: t|| = ||h|| + ||t|| + 1 \quad ||\mathtt{f}\, x|| = ||x||$$

This assignment satisfies $||l|| \geq ||r||$ for any $l \to r \in \mathcal{R}$, and also satisfies the definition of an additive and polynomial *quasi-interpretation* [12]. Such definition, along with a termination condition from a product path ordering, implies that this STTRS terminates in linear time and uses linear space, i.e. it computes a function in `FPTIME`. In particular, as `len` has a fixed size, it will also terminate in linear time by our translation result.

Example 12 (Higher-order). We finish this section by giving an example of higher-order functions. We can write down the standard `map` function between list in `Hyrql` as:

$$\mathtt{map} = \mathtt{letrec}\, f\, \phi = \lambda x.\mathtt{match}\, x\, \{[\,] \to [\,], h :: t \to (\phi\, h) :: ((f\, \phi) t)\}$$

Which can then be translated into the following STTRS:

$$\{\mathtt{map}(\phi, [\,]) \to [\,], \quad \mathtt{map}(\phi, h :: t) \to (\phi\, h) :: (\mathtt{map}(\phi, t))\}$$

Such STTRS have been proven to terminate with a polynomial time bound under reasonable assumption for `f` [8] (i.e. `f` being given an additive polynomial interpretation).

Remark that if one gives a quantum program with an associated quantum circuit, properties on the circuit can arise from such results. Indeed, termination of the quantum program implies that the circuit uses a finite number of unitary gates, and is thus finite. Furthermore, the obtained complexity bounds can translate in bounds on the number of used gates; in particular, time and space bounds can both relate to the depth and width of the quantum circuit.

5 Conclusion

We presented `Hyrql`, a hybrid quantum functional language with general recursion. It comes equipped with a type system and call-by-value operational semantics, and enjoys the standard properties of programming languages. We also designed an algorithm to translate any term of the language into a semantically equivalent simply-typed term rewrite system, and we proved that the reduction length for an `Hyrql` program is the same as the runtime complexity of its translation up to a polynomial factor.

As discussed during Section 3, our type system is undecidable due to the orthogonality and unitarity general definition. Orthogonality could be adapted to feature more decidable cases; for example, in the Quantum Switch defined in Example 2, both elements in the `qcase` construct will always be orthogonal as long as f, g terminate. We could also restrict our language to use a similar criterion to `Spm` [45] and obtain a decidable criterion for unitarity.

As we have shown through different examples, our framework is generic as it is capable of using existing techniques to provide termination and complexity bounds. Such reasoning could be generalized to specific term subsets that would ensure space or time complexity bounds. Furthermore, our algorithm is defined by splitting a program into multiple small subprograms. Assuming we have a purely quantum program, this gives a way to interpret subprograms as quantum gates, thus transpiling terms into quantum circuits. Refining existing techniques for complexity would make it possible to give bounds on depth and width of such quantum circuits.

References

1. Abbott, A.A., Wechs, J., Horsman, D., Mhalla, M., Branciard, C.: Communication through coherent control of quantum channels. Quantum **4**, 333 (Sep 2020). https://doi.org/10.22331/q-2020-09-24-333
2. Altenkirch, T., Grattage, J.: A functional quantum programming language. In: Logic in Computer Science, LICS'05. pp. 249–258. IEEE Computer Society (2005). https://doi.org/10.1109/lics.2005.1
3. Araújo, M., Costa, F., Brukner, v.: Computational advantage from quantum-controlled ordering of gates. Phys. Rev. Lett. **113**, 250402 (Dec 2014). https://doi.org/10.1103/PhysRevLett.113.250402
4. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science **236**(1), 133–178 (2000). https://doi.org/10.1016/S0304-3975(99)00207-8

5. Avanzini, M., Moser, G.: Dependency pairs and polynomial path orders. In: International Conference on Rewriting Techniques and Applications, RTA'09. pp. 48–62. Springer (2009). https://doi.org/10.1007/978-3-642-02348-4_4
6. Avanzini, M., Moser, G., Péchoux, R., Perdrix, S., Zamdzhiev, V.: Quantum expectation transformers for cost analysis. In: Baier, C., Fisman, D. (eds.) LICS '22: Symposium on Logic in Computer Science. pp. 10:1–10:13. ACM (2022). https://doi.org/10.1145/3531130.3533332
7. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (Mar 1998). https://doi.org/10.1017/cbo9781139172752
8. Baillot, P., Dal Lago, U.: Higher-order interpretations and program complexity. Information and Computation **248**, 56–81 (2016). https://doi.org/10.1016/j.ic.2015.12.008
9. Barendregt, H.: The lambda calculus: its syntax and semantics. Studies in logic and the foundations of Mathematics (1984)
10. Barsse, K., Péchoux, R., Perdrix, S.: A quantum programming language for coherent control (2025), https://arxiv.org/abs/2507.10466
11. Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. Theoretical Computer Science **560**, 7–11 (2014). https://doi.org/10.1016/j.tcs.2014.05.025
12. Bonfante, G., Marion, J.Y., Moyen, J.Y.: Quasi-interpretations a way to control resources. Theoretical Computer Science **412**(25), 2776–2796 (Jun 2011). https://doi.org/10.1016/j.tcs.2011.02.007
13. Bădescu, C., Panangaden, P.: Quantum alternation: Prospects and problems. Electronic Proceedings in Theoretical Computer Science **195**, 33–42 (Nov 2015). https://doi.org/10.4204/eptcs.195.3
14. Chardonnet, K., Lemonnier, L., Valiron, B.: Semantics for a Turing-Complete Reversible Programming Language with Inductive Types. In: Rehof, J. (ed.) Formal Structures for Computation and Deduction (FSCD 2024). LIPIcs, vol. 299, pp. 19:1–19:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPIcs.FSCD.2024.19
15. Chardonnet, K., Saurin, A., Valiron, B.: A Curry-Howard Correspondence for Linear, Reversible Computation. LIPIcs, Volume 252, CSL 2023 **252**, 13:1–13:18 (2023). https://doi.org/10.4230/LIPICS.CSL.2023.13
16. Chareton, C., Bardin, S., Bobot, F., Perrelle, V., Valiron, B.: An automated deductive verification framework for circuit-building quantum programs. In: Yoshida, N. (ed.) Programming Languages and Systems, 30th European Symposium on Programming, ESOP 2021. pp. 148–177. Springer (2021). https://doi.org/10.1007/978-3-030-72019-3_6
17. Chiribella, G., D'Ariano, G.M., Perinotti, P., Valiron, B.: Quantum computations without definite causal structure. Phys. Rev. A **88**, 022318 (Aug 2013). https://doi.org/10.1103/PhysRevA.88.022318
18. Colledan, A., Dal Lago, U.: Flexible type-based resource estimation in quantum circuit description languages. Proc. ACM Program. Lang. **9**(POPL) (Jan 2025). https://doi.org/10.1145/3704883
19. Cross, A., Javadi-Abhari, A., Alexander, T., De Beaudrap, N., Bishop, L.S., Heidel, S., Ryan, C.A., Sivarajah, P., Smolin, J., Gambetta, J.M., Johnson, B.R.: Openqasm 3: A broader and deeper quantum assembly language. ACM Transactions on Quantum Computing **3**(3), 1–50 (Sep 2022). https://doi.org/10.1145/3505636
20. Cross, A.W., Bishop, L.S., Smolin, J.A., Gambetta, J.M.: Open quantum assembly language (2017), https://arxiv.org/abs/1707.03429

21. Dal Lago, U., Masini, A., Zorzi, M.: Quantum implicit computational complexity. Theor. Comput. Sci. **411**(2), 377–409 (2010). https://doi.org/10.1016/J.TCS.2009.07.045
22. Dave, K., Lemonnier, L., Péchoux, R., Zamdzhiev, V.: Combining quantum and classical control: syntax, semantics and adequacy. In: Foundations of Software Science and Computation Structures, FoSSaCS 2025. p. 155–175. Springer-Verlag, Berlin, Heidelberg (2025). https://doi.org/10.1007/978-3-031-90897-2_8
23. Dershowitz, N.: Orderings for term-rewriting systems. Theoretical computer science **17**(3), 279–301 (1982). https://doi.org/10.1016/0304-3975(82)90026-3
24. Dershowitz, N.: Termination of rewriting. J. Symb. Comput. **3**(1–2), 69–115 (Feb 1987). https://doi.org/10.1016/S0747-7171(87)80022-6
25. Díaz-Caro, A., Malherbe, O.: Quantum control in the unitary sphere: Lambda-s1 and its categorical model. Log. Methods Comput. Sci. **18**(3) (2022). https://doi.org/10.46298/LMCS-18(3:32)2022
26. Endrullis, J., Geuvers, H., Zantema, H.: Degrees of Undecidability in Term Rewriting, p. 255–270. Springer Berlin Heidelberg (2009). https://doi.org/10.1007/978-3-642-04027-6_20
27. Feng, Y., Ying, M.: Quantum hoare logic with classical variables. ACM Transactions on Quantum Computing **2**(4), 1–43 (2021). https://doi.org/10.1145/3456877
28. Ferrari, F., Hainry, E., Péchoux, R., Silva, M.: Quantum programming in polylogarithmic time. In: Mathematical Foundations of Computer Science, MFCS 2025. LIPIcs, vol. 345, pp. 47:1–47:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2025). https://doi.org/10.4230/LIPICS.MFCS.2025.47
29. Green, A.S., Lumsdaine, P.L., Ross, N.J., Selinger, P., Valiron, B.: Quipper: a scalable quantum programming language. In: ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13. p. 333–342. ACM (Jun 2013). https://doi.org/10.1145/2491956.2462177
30. Hainry, E., Péchoux, R., Silva, M.: A programming language characterizing quantum polynomial time. In: Foundations of Software Science and Computation Structures, Fossacs 2023. pp. 156–175. Springer (Apr 2023). https://doi.org/10.1007/978-3-031-30829-1_8
31. Hainry, E., Péchoux, R., Silva, M.: Branch sequentialization in quantum polytime. In: Fernández, M. (ed.) Formal Structures for Computation and Deduction, FSCD 2025. LIPIcs, vol. 337, pp. 22:1–22:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2025). https://doi.org/10.4230/LIPICS.FSCD.2025.22
32. Halava, V., Harju, T., Hirvensalo, M., Karhumäki, J.: Skolem's problem - on the border between decidability and undecidability. Tech. Rep. 683, Turku Center for Computer Science (2005), http://www.tucs.fi/publications/insight.php?id=tHaHaHiKa05a
33. Knill, E.: Conventions for quantum pseudocode. arXiv preprint arXiv:2211.02559 (2022)
34. Kristjánsson, H., Odake, T., Yoshida, S., Taranto, P., Bavaresco, J., Quintino, M.T., Murao, M.: Exponential separation in quantum query complexity of the quantum switch with respect to simulations with standard quantum circuits (2024), https://arxiv.org/abs/2409.18420
35. Lafont, Y.: Soft linear logic and polynomial time. Theor. Comput. Sci. **318**(1-2), 163–180 (2004). https://doi.org/10.1016/J.TCS.2003.10.018
36. Lankford, D.S.: On proving term rewriting systems are noetherien. Tech. rep., Department of Mathematics, Louisiana technical University (1979), https://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford_Poly_Term.pdf

37. Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: Symposium on Principles of Programming Languages, POPL 2001. p. 81–92. ACM, New York, NY, USA (Jan 2001). https://doi.org/10.1145/373243.360210
38. Lemonnier, L.: The Semantics of Effects : Centrality, Quantum Control and Reversible Recursion. Ph.D. thesis, Université Paris-Saclay (Jun 2024), https://theses.hal.science/tel-04625771
39. Marion, J.Y., Péchoux, R.: Sup-interpretations, a semantic method for static analysis of program resources. ACM Trans. Comput. Logic **10**(4) (Aug 2009). https://doi.org/10.1145/1555746.1555751
40. Moser, G.: Proof Theory at Work: Complexity Analysis of Term Rewrite Systems. Cumulative habilitation thesis, University of Innsbruck (2009), http://arxiv.org/abs/0907.5527
41. Nies, A.: Computability and Randomness. Oxford University Press (01 2009). https://doi.org/10.1093/acprof:oso/9780199230761.001.0001
42. Péchoux, R.: Complexité implicite : bilan et perspectives. Accreditation to supervise research, Université de Lorraine (Oct 2020), https://hal.univ-lorraine.fr/tel-02978986
43. Procopio, L.M., Moqanaki, A., Araújo, M., Costa, F., Alonso Calafell, I., Dowd, E.G., Hamel, D.R., Rozema, L.A., Brukner, v., Walther, P.: Experimental superposition of orders of quantum gates. Nature Communications **6**(1) (Aug 2015). https://doi.org/10.1038/ncomms8913
44. Sabry, A., Valiron, B., Vizzotto, J.K.: From symmetric pattern-matching to quantum control. In: Baier, C., Dal Lago, U. (eds.) Foundations of Software Science and Computation Structures. pp. 348–364. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-89366-2_19
45. Sabry, A., Valiron, B., Vizzotto, J.K.: From Symmetric Pattern-Matching to Quantum Control. In: Baier, C., Dal Lago, U. (eds.) Foundations of Software Science and Computation Structures. pp. 348–364. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-89366-2_19
46. Selinger, P.: Towards a quantum programming language. Mathematical Structures in Computer Science **14**(4), 527–586 (Aug 2004). https://doi.org/10.1017/s0960129504004256
47. Selinger, P., Valiron, B.: A lambda calculus for quantum computation with classical control. In: Urzyczyn, P. (ed.) Typed Lambda Calculi and Applications, TLCA 2005. pp. 354–368. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11417170_26
48. Selinger, P., Valiron, B.: Semantic techniques in quantum computation. In: Gay, S., Mackie, I. (eds.) Quantum lambda calculus, pp. 135–172. Cambridge University Press (2009). https://doi.org/10.1017/cbo9781139193313.005
49. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172
50. Taddei, M.M., Cariñe, J., Martínez, D., García, T., Guerrero, N., Abbott, A.A., Araújo, M., Branciard, C., Gómez, E.S., Walborn, S.P., Aolita, L., Lima, G.: Computational advantage from the quantum superposition of multiple temporal orders of photonic gates. PRX Quantum **2**, 010320 (Feb 2021). https://doi.org/10.1103/PRXQuantum.2.010320
51. Voichick, F., Li, L., Rand, R., Hicks, M.: Qunity: A unified language for quantum and classical computing. Proc. ACM Program. Lang. **7**(POPL) (Jan 2023). https://doi.org/10.1145/3571225

52. Yamada, T.: Confluence and Termination of Simply Typed Term Rewriting Systems. In: Middeldorp, A. (ed.) Rewriting Techniques and Applications. pp. 338–352. Springer, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-45127-7_25
53. Ying, M.: Foundations of quantum programming. Elsevier (2024). https://doi.org/10.1016/C2014-0-02660-3

A Additional Material for Section 2

For the sake of simplicity in the following typing trees, we maye denote $;\vdash t : T$ for $\varnothing;\varnothing \vdash t : T$.

Example 13. As $|0\rangle, |1\rangle$ are pure values, then we get $|0\rangle \perp |1\rangle$ directly from the definition. This allows us to type $|\pm\rangle$:

$$\dfrac{\dfrac{}{;\vdash |0\rangle : \mathrm{Qbit}}\,(\mathrm{ax}_0) \quad \dfrac{}{;\vdash |1\rangle : \mathrm{Qbit}}\,(\mathrm{ax}_1) \quad |\tfrac{1}{\sqrt{2}}|^2 + |\pm\tfrac{1}{\sqrt{2}}|^2 = 1 \quad |0\rangle \perp |1\rangle}{;\vdash |\pm\rangle : \mathrm{Qbit}}\,(\mathrm{sup})$$

It is also easy to derive $|+\rangle \perp |-\rangle$:

$$\begin{aligned}\langle |+\rangle, |-\rangle\rangle &= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}\delta_{|0\rangle,|0\rangle} - \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}\delta_{|0\rangle,|1\rangle} + \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}\delta_{|1\rangle,|0\rangle} - \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}\delta_{|1\rangle,|1\rangle}\\ &= \frac{1}{2} - \frac{1}{2} = 0\end{aligned}$$

Using $h = \lambda x.\mathtt{qcase}\,x\left\{|0\rangle \to |+\rangle, |1\rangle \to |-\rangle\right\}$ for conciseness, we can type the Hadamard gate as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing; x : \mathrm{Qbit} \vdash x : \mathrm{Qbit}} \quad ;\vdash |+\rangle : \mathrm{Qbit} \quad ;\vdash |-\rangle : \mathrm{Qbit} \quad |+\rangle \perp |-\rangle}{\varnothing; x : \mathrm{Qbit} \vdash \mathtt{qcase}\,x\left\{|0\rangle \to |+\rangle, |1\rangle \to |-\rangle\right\} : \mathrm{Qbit}}\,(\mathrm{qcase})}{;\vdash : \mathrm{Qbit} \multimap \mathrm{Qbit}}\,(\mathrm{lbd}) \quad h \text{ is unitary}}{;\vdash \mathtt{Had} : \mathrm{Qbit} \leftrightarrow \mathrm{Qbit}}\,(\mathrm{unit})$$

The fact that h is unitary is direct from the definition: it is surjective, as for any $v = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$, take $w = \alpha \cdot |+\rangle + \beta \cdot |-\rangle$; the fact that it is an isometry can be derived easily as $|+\rangle \perp |-\rangle$, or by using Proposition 3.

Example 14. The quantum switch from Example 2 can be typed by our language. For conciseness, we write $v = |0\rangle \otimes f(g\,t)$, $w = |1\rangle \otimes g(f\,t)$, $\Gamma = f : \mathrm{Qbit} \leftrightarrow \mathrm{Qbit}, g : \mathrm{Qbit} \leftrightarrow \mathrm{Qbit}$, $s = \mathtt{match}\,x\left\{(c \otimes t)\mathtt{qcase}\,c\left\{|0\rangle \to v, |1\rangle \to w\right\}\right\}$, $\Delta = c : \mathrm{Qbit}, t : \mathrm{Qbit}$ and $Q_2 = \mathrm{Qbit} \otimes \mathrm{Qbit}$. Note that $v \perp w$, as f, g terminate as they are unitary, thus both reducing to a qubit, and orthogonality derives from the first qubit.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma; c : \text{Qbit} \vdash c : \text{Qbit} \quad \Gamma; t : \text{Qbit} \vdash v : Q_2 \quad \Gamma; t : \text{Qbit} \vdash w : Q_2 \quad v \perp w}{\Gamma; \Delta \vdash \texttt{qcase}\, c\, \{|0\rangle \to v\,, |1\rangle \to w\} : Q_2} \qquad \Gamma; x : Q_2 \vdash x : Q_2}{\Gamma; x : Q_2 \vdash s : Q_2}}{\Gamma; x : Q_2 \vdash \lambda x.s : Q_2 \multimap Q_2}}{f : Q_2 \leftrightarrow Q_2; \varnothing \vdash \lambda g.\lambda x.s : (\text{Qbit} \leftrightarrow \text{Qbit}) \Rightarrow (Q_2 \multimap Q_2)}}{; \vdash \texttt{QS} : (\text{Qbit} \leftrightarrow \text{Qbit}) \Rightarrow (\text{Qbit} \leftrightarrow \text{Qbit}) \Rightarrow (Q_2 \multimap Q_2)}$$

Example 15. Coming back on Example 8, the following typing tree derives typing for `len`, for a given type $A \in \mathbb{B}_C$, with the following notations to shorten the tree: $\Gamma_f = f : [A] \Rightarrow \texttt{nat}$, $\Gamma_x = x : [A]$, and $\Gamma = \Gamma_f, \Gamma_x, h : A, t : [A]$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_f, \Gamma_x; \varnothing \vdash x : [A]} \quad \dfrac{\texttt{zero} :: \texttt{nat}}{\Gamma_f, \Gamma_x; \varnothing \vdash 0 : \texttt{nat}}\,(\text{cons}) \quad \dfrac{S :: \texttt{nat} \to \texttt{nat} \qquad \dfrac{\dfrac{}{\Gamma; \varnothing \vdash f : [A] \Rightarrow \texttt{nat}}\,(\text{ax}_c) \quad \dfrac{}{\Gamma; \varnothing \vdash t : [A]}\,(\text{ax}_c)}{\Gamma; \varnothing \vdash f t : \texttt{nat}}\,(\text{app}_c)}{\Gamma; \varnothing \vdash S(f t) : \texttt{nat}}\,(\text{cons})}{\Gamma_f, \Gamma_x; \varnothing \vdash \texttt{match}\, x\, \{[\,] \to 0\,, h :: t \to S(f t)\} : \texttt{nat}}\,(\text{match})}{\Gamma_f; \varnothing \vdash \lambda x.\texttt{match}\, x\, \{[\,] \to 0\,, h :: t \to S(f t)\} : [A] \Rightarrow \texttt{nat}}\,(\text{abs}_c)}{; \vdash \texttt{letrec}\, f\, x = \texttt{match}\, x\, \{[\,] \to 0\,, h :: t \to S(f t)\} : [A] \Rightarrow \texttt{nat}}\,(\text{fix})$$

In particular, as h is not present in the syntax of $S(f\,t)$, we could not type `len` if h must be used linearly, thus if $A \in Q$.

However, using `shape`, one can type $; \vdash \lambda y.(y, \texttt{shape}(y)) : \text{Qbit} \multimap [\text{Qbit}] \times \texttt{nat}$. Again, to shorten the tree, we use $\Delta_t = t : [\text{Qbit}]$ for any t, $\texttt{pr} = \texttt{pair}_{[\text{Qbit}],\texttt{nat}}$, and as $\texttt{shape}([\text{Qbit}]) = [\mathbb{1}]$:

$$\dfrac{\dfrac{\dfrac{\texttt{pr} :: [\text{Qbit}], \texttt{nat} \to [\text{Qbit}] \times \texttt{nat} \quad [\Delta_{t'}]; \Delta_t \vdash t : [\text{Qbit}] \qquad \dfrac{\dfrac{\varnothing; \Delta_{t'} \vdash t' : [\text{Qbit}]}{[\Delta_{t'}]; \varnothing \vdash \texttt{shape}(t') : [\mathbb{1}]}\,(\text{shape}) \quad [\Delta_{t'}]; \varnothing \vdash \texttt{len}_{\mathbb{1}} : [\mathbb{1}] \Rightarrow \texttt{nat}}{[\Delta_t]; \varnothing \vdash \texttt{len}_{\mathbb{1}}\texttt{shape}(t') : \texttt{nat}}\,(\text{app}_c)}{[\Delta_{t'}]; \Delta_t \vdash (t, \texttt{shape}(t')) : [\text{Qbit}] \times \texttt{nat}}\,(\text{cons})}{\varnothing; \Delta_t \vdash (t, \texttt{shape}(t)) : [\text{Qbit}] \times \texttt{nat}}\,(\text{contr})}{; \vdash \lambda t.(t, \texttt{shape}(t)) : [\text{Qbit}] \multimap [\text{Qbit}] \times \texttt{nat}}\,(\text{abs})$$

Example 16. Assuming we have a bit type `bit` for both constructors $0 :: \texttt{bit}$ and $1 :: \texttt{bit}$, the typing of Example 4 can be derived, by combining most of the techniques exhibited from the previous examples, giving us $;\vdash$ `keygen` : `list`(`bit`) $\Rightarrow$ `list`(Qbit). Note that when we used programs in the syntax of other terms, such as `Had`, it is just notation to avoid rewriting the whole term.

B Proofs of Section 3

B.1 Intermediate results

In this subsection, we state some intermediate results and definitions that we will use through the main proofs. First, we denote $\mathcal{T}, \Lambda, \Lambda_{\vdash}$ respectively for the set of types, terms, and well-typed terms of the language. Furthermore, we may use the notation $\{x \to v\}$ for a substitution $\{v/x\}$, and define the *support* of a substitution $\mathrm{Supp}(\sigma)$ as the set of variables x_i such that there exists t_i with $\{t_i/x_i\} \in \sigma$.

Lemma 1 (Weakening). *Let $\Gamma; \Delta \vdash t : T$ be a well-typed term. Then $\Gamma, \Gamma'; \Delta \vdash t : T$ holds for any context Γ' compatible with Γ, Δ.*

Proof. This can be proven by induction on t, as the axiom rules satisfy this weakening property. Furthermore, the only variable removed from the non-linear context are bound variables, thus Γ' stays untouched at each point. □

Lemma 5. *Let $T \in \mathcal{T}$. Then, there exists $n \in \mathbb{N}$, $T_i \in \mathcal{T}$ for $1 \leq i \leq n$, and a base type $B^{|\rangle}$ such that $T = T_1 \rightarrowtail \ldots \rightarrowtail T_n \rightarrowtail B^{|\rangle}$.*

Proof. Direct by induction on T: either it is a base type itself, thus we take $n = 0$; or $T = T_1 \rightarrowtail T'$, then we use the induction hypothesis on T' and right associativity to conclude. □

We also introduce an operator $\theta_p(t)$, called the *quantity*, which aims to measure the amplitude of a pure term p in a term t, e.g. a superposition.

Definition 6. *Let p be a pure term. We define the following map $\theta_p : \Lambda \to \mathbb{C}$ as follows:*

$$\theta_p(x) \triangleq \delta_{p,x} \qquad \theta_p(|0\rangle) \triangleq \delta_{p,|0\rangle} \qquad \theta_p(|1\rangle) \triangleq \delta_{p,|1\rangle}$$

$$\theta_p(\mathtt{qcase}\, t' \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\}) \triangleq \delta_{p,\mathtt{qcase}\, p' \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\}} \theta_{p'}(t')$$

$$\theta_p(c(t_1, \ldots, t_n)) \triangleq \delta_{p,c(p_1,\ldots,p_n)} \times_{i=1}^{n} \theta_{p_i}(t_i)$$

$$\theta_p(\mathtt{match}_{1\leq i\leq n}\, t' \,\{c_i(\overrightarrow{x_i}) \to t_i\}) \triangleq \delta_{p,\mathtt{match}_{1\leq i\leq n}\, p' \,\{c_i(\overrightarrow{x_i})\to t_i\}} \theta_{p'}(t')$$

$$\theta_p(\lambda x.t) \triangleq \delta_{p,\lambda x.t} \qquad \theta_p(\mathtt{letrec}\, f\, x = t) \triangleq \delta_{p,\mathtt{letrec}\, f\, x=t}$$

$$\theta_p(\mathtt{unit}(t)) \triangleq \delta_{p,\mathtt{unit}(t)} \qquad \theta_p(t_1 t_2) \triangleq \delta_{p,p_1 p_2} \nu_{p_1,t_1} \nu_{p_2,t_2}$$

$$\theta_p(\textstyle\sum_{i=1}^{n} \alpha_i \cdot t_i) \triangleq \textstyle\sum_{i=1}^{n} \alpha_i \cdot \theta_p(t_i) \qquad \theta_p(\mathtt{shape}(t')) \triangleq \delta_{p,\mathtt{shape}(p')} \nu_{p',t'}$$

We have used the notation $\delta_{p,t}$ for the Kronecker symbol, and $\nu_{p,t}$ for the equivalence Kronecker symbol, meaning $\nu_{p,t} = 1$ iff $p \equiv t$.

Lemma 6. *Let p be a pure term. Then $\theta_p(p) = 1$.*

Proof. Proven by induction on p, direct by definition of the quantity. □

Lemma 7. *Let p be a pure term and t, t' be two terms of the language. If $t \equiv t'$, then $\theta_p(t) = \theta_p(t')$.*

Proof. By induction on each rule defining $\equiv$ in Table 3; the transitive and reflexive case can be derived from this. The first two lines of rules are verified directly as $\mathbb{C}$ is a vectorial space. The three linearity rules also work by definition of summation and the linear constructs. Finally, we can prove the $C_\equiv$ case by doing an induction on the depth of $C_\equiv$, the base case being just $t \equiv t'$ and then obtaining the result by induction. □

Lemma 8. *Let s, t be two pure terms. Then $\theta_s(t) = \nu_{s,t}$. Furthemore, if s, t are values, then $\theta_s(t) = \delta_{s,t}$.*

Proof. Note by Lemma 6, $\theta_s(s) = 1$. If $\theta_s(t) = 0$, then $s \not\equiv t$, else Lemma 7 would fail. Now, one can check by induction on t that if $\theta_s(t) = 1$, then $s \equiv t$. This is done directly, mainly proving $\equiv$ using $C_\equiv$. The second part of the lemma is also obtained directly by induction of t, by looking at the definition of the quantity. □

Lemma 9. *Let t be a term of the language. Suppose $t \not\equiv 0 \cdot t'$. Then t has a unique canonical form $\sum_{i=1}^{n} \alpha_i \cdot t_i$, up to commutativity and equivalence on t_i.*

Proof. Let us prove existence by induction on t.

– If $t = x, |0\rangle, |1\rangle, \lambda x.s, \texttt{letrec}\, f\, x = s, \texttt{unit}(s), s_1 s_2$ or $\texttt{shape}(s)$, then t is directly pure and $t \equiv 1 \cdot t$, which is a canonical form.
– Suppose $t = c(t_1, \ldots, t_n)$. If we have any $t_i \equiv 0 \cdot t_i'$, then $t \equiv 0 \cdot c(t_1, \ldots, t_i', \ldots, t_n)$. Therefore, we can apply the induction hypothesis on each t_i, meaning we have $t_i \equiv \sum_{j_i=1}^{n_i} \alpha_{j_i} \cdot t_{j_i}$. By developing t under $\equiv$, we have $t \equiv \sum_{j_1=1}^{n_1} \cdots \sum_{j_n=1}^{n_n} \alpha_{j_1} \ldots \alpha_{j_n} \cdot c(t_{j_1}, \ldots, t_{j_n})$. This can be seen as one big sum $\sum_{k=1}^{n_1 \ldots n_k} \alpha_k \cdot c(t_1^k, \ldots, t_n^k)$, where $\alpha_k = \alpha_{j_1} \ldots \alpha_{j_n}$. In particular, $\alpha_k \neq 0$ as each $\alpha_{j_i} \neq 0$; and if $c(t_1^k, \ldots, t_n^k) \equiv c(t_1^{k'}, \ldots, t_n^{k'})$, then by Lemma 7, $\theta_{c(t_1^k, \ldots, t_n^k)}(c(t_1^{k'}, \ldots, t_n^{k'})) = 1$ as they are both pure, but by analyzing the formula, this implies $\theta_{t_i^k}(t_i^{k'}) = 1$, thus by Lemma 7, $t_i^k \equiv t_i^{k'}$. By induction hypothesis, this is not possible for all $1 \leq i \leq n$, therefore terms are pairwise not equivalent, which concludes.
– The same process can be done for $t = \texttt{qcase}\, t\, \{\ldots\}$ or $t = \texttt{match}\, t\, \{\ldots\}$, as they have an identical behaviour to a constructor with one slot.
– Finally, suppose $t = \sum_{i=1}^{n} \alpha_i \cdot t_i$. Any $t_i \equiv 0 \cdot t_i'$ will be removed from the sum through $t + 0 \cdot t' \equiv t$, then we can apply the i.h. on the remaining t_i. Finally, using distributivity, we can obtain a big sum $\sum_{k=1}^{n} \beta_k \cdot s_k$, where s_k are pure terms. We can group s_k that are equivalent, then remove any potential term with a 0 phase, and the term obtained satisfies the canonical form definition. Note that as $t \not\equiv 0 \cdot t'$, the sum obtained is actually well defined, in particular there is at least one term.

Unicity comes from our quantity function defined above. Suppose t has two canonical forms $t \equiv \sum_{i=1}^{n} \alpha_i \cdot t_i \equiv \sum_{j=1}^{m} \beta_j \cdot s_j$. As all t_i are pure, for any $1 \leq i_0 \leq n$, $\theta_{t_{i_0}}(\sum_{i=1}^{n} \alpha_i \cdot t_i) = \sum_{i=1}^{n} \alpha_i \theta_{t_{i_0}}(t_i)$. By Lemma 8, $\theta_{t_{i_0}}(t_i) = \nu_{t_{i_0}, t_i}$, however the t_i are not equivalent pairwise, thus $\theta_{t_{i_0}}(\sum_{i=1}^{n} \alpha_i \cdot t_i) = \alpha_{i_0} \neq 0$. However, by the same argument, $\theta_{t_{i_0}}(\sum_{j=1}^{m} \beta_j \cdot s_j) = \sum_{j=1}^{m} \beta_j \theta_{t_{i_0}}(s_j)$. If there is no $s_j \equiv t_{i_0}$, then Lemma 7 fails, thus there is at least one; and there cannot be more than one, else, $s_j \equiv t_{i_0} \equiv s_{j'}$, thus by transitivity $s_j \equiv s_{j'}$, which is not possible as this is a canonical form. Therefore, there exists j_0 such that $\theta_{t_{i_0}}(\sum_{j=1}^{m} \beta_j \cdot s_j) = \beta_{j_0}$; by Lemma 7, we obtain $\alpha_{i_0} = \beta_{j_0}$. Thus, any t_i has a unique and distinct (as they are not equivalent) match in s_j, therefore $n \leq m$. The same process can be done the other way around, so $m \leq n$, thus $m = n$. So each sum contains the same number of elements, such that there is a map from $1, \ldots, n$ to $1, \ldots, n$ that associates i with j such that $t_i \equiv s_j$ and $\alpha_i = \beta_j$. Therefore, both sums are equal up to an equivalence on t_i and permutation of the t_i. □

Lemma 10. *Let t be a well-typed term, suppose that t terminates and has a canonical form $\sum_{i=1}^{n} \alpha_i \cdot s_i$. Then any element has the same shape, meaning* $\texttt{shape}(s_i) \rightsquigarrow^* v_i$, $\texttt{shape}(s_j) \rightsquigarrow^* v_j$, *and $v_i = v_j$ for any i, j.*

Proof. By induction on the typing, and the constructed canonical form. Few points to note:

- for (qcase), as $t_0 \perp t_1$, in particular they have the same shape, thus any s_i will reduce to a superposition of t_0, t_1;
- for (match), even if the inside argument is the superposition, by induction hypothesis, all elements have the same shape, thus in particular reduce to the same t_i.
- For (sup), $t_i \equiv t_j$, thus they have the same shape. □

Another process that we will do in proofs is completing canonical forms. Given two terms s, t with canonical forms $\sum_{i=1}^{n} \alpha_i \cdot s_i$ and $\sum_{j=1}^{m} \beta_j \cdot t_j$, one can always create the set S, that contains all the s_i, t_j such that they are pairwise non equivalent. Then, by adding $0 \cdot t'$, we can write each canonical form as $\sum_{k=1}^{l} \alpha_l \cdot w_l$ and $\sum_{k=1}^{l} \beta_l \cdot w_l$. This is not a canonical form as some phases are zero, but it is equivalent to the first canonical form, thus to s, t; and such shape is actually close enough to keep most existing results on the canonical form, such as normalization.

The following lemma abuses the operational semantics, by supposing that we have a rule allowing to reduce any term $\sum_{i=1}^{n} 0 \cdot t_i$ to $\sum_{i=1}^{n} 0 \cdot t'_i$ when $t_i \rightsquigarrow^? t'_i$. Our type system can easily be expanded to satisfy the properties; in fact, all the following results, i.e. subject reduction and confluence are proven by taking this case into account. Furtheremore, we will later prove that any well-typed term cannot be expressed as so, thus this rule is only necessary for the proofs, and we have thus chosen to omit it.

Lemma 11. *Let $t = \sum_{i=1}^{n} \alpha_i \cdot t_i$, suppose $t_i \rightsquigarrow^* v_i$. Then $t \rightsquigarrow^* \sum_{i=1}^{n} \alpha_i \cdot v_i$.*

Proof. For each t_i such that $t_i \equiv 0 \cdot s$, or such that $\alpha_i = 0$, we can remove it via (equiv), thus $t \equiv \sum_{k=1}^{l} \alpha_k \cdot t_k$. By writing each canonical form, up to completion, as $t_k \equiv \sum_{j=1}^{m} \beta_{k,j} \cdot s_j$, $s_j \rightsquigarrow^? s'_j$ and $\sum_{k=1}^{l} \alpha_k \cdot \beta_{k,j} = \gamma_j$:

$$t \equiv \sum_{j=1}^{m} \gamma_j \cdot s_j \equiv \sum_{j,\gamma_j \neq 0} \gamma_j \cdot s_j \rightsquigarrow \sum_{j,\gamma_j \neq 0} \gamma_j \cdot s'_j \equiv \sum_{j=1}^{m} \gamma_j \cdot s'_j \equiv \sum_{k=1}^{n} \alpha_k \cdot t'_k,$$

as the s_j are pairwise not equivalent, either all $\gamma_j = 0$, or it is a canonical form. If the obtained form is a value, then we can stop here, and the result will be correct; even if t_i is not a value, the non-value parts will still be canceled via (Equiv) at the end, as the reduction is deterministic. Else, we can appy (Can), and we can replace $0 \cdot s_j$ by $0 \cdot s'_j$. Furthermore, we can add back the removed terms from the beginning, adding back t'_i instead of t_i, to get that $\sum_{k=1}^{l} \alpha_k \cdot t'_k \equiv \sum_{i=1}^{n} \alpha_i \cdot t'_i$. We conclude by (Equiv), and chaining this multiple times; such process terminates as the t_i do terminate. ☐

C Proofs of Section 3

C.1 Proofs of confluence

Lemma 12 (Equivalence preservation). *Let s_1, s_2 be two well-typed terms. Suppose $s_i \rightsquigarrow t_i$ for $i = 1, 2$. Then $t_1 \equiv t_2$.*

Proof. We prove this result by induction of the sum of the depth of the derivation trees of $s_i \rightsquigarrow t_i$.

Consider the base case where both derivation trees have the same root. Apart from (Can) and (Shape$_s$), having the same root implies that $s_1 = s_2$, and as reduction is deterministic, $t_1 = t_2$, thus $t_1 \equiv t_2$. In the case of (Can), by Lemma 9, if one has a canonical form, then by $\equiv$ the second also has a canonical form, which is identical up to commutation and equivalence on the terms. Thus, $s_1 = \sum_{j=1}^{n} \alpha_i \cdot r_i$, and $s_2 = \sum_{j=1}^{n} \beta_i \cdot q_i$, where both sums are identical, up to commutation and each q_i is equivalent to a r_j. We can apply the induction hypothesis on q_i, r_j, to get that the reduced terms are also equivalent, and up to commuting the terms back, we obtain the equivalence at the end. If not, both terms have 0 phase, and equivalence is direct from this. The same process can be applied for (Shape$_s$), but inside the `shape` construct, by Lemma 10, as it is a value.

Now, suppose that the trees do not have the same root. First, do not consider any (Shape) rule. Again, apart from (Can) and (Equiv), all left terms are pure, and by Lemma 7, no distinct rule can yield two equivalent terms. Suppose the first tree has a (Equiv) root, meaning:

$$\frac{s_1 \equiv q_1 \qquad q_1 \rightsquigarrow r_1 \qquad r_1 \equiv t_1}{s_1 \rightsquigarrow t_1}\ \text{(Equiv)}$$

By transitivity, $q_1 \equiv s_2$, thus we can apply the induction hypothesis on $q_1 \rightsquigarrow r_1$ and $s_2 \rightsquigarrow t_2$ to get $r_1 \equiv t_2$, and by transitivity, $t_1 \equiv t_2$. Same can be done for the right term.

Now, suppose the first tree has a (Can) root. As the right tree has neither a (Can) or (Equiv) root, it has a root with a a pure term. This implies that all phases are not zero, and that s_1 is a canonical form. By uniqueness, $s_1 = 1 \cdot q_1$, with $q_1 \equiv s_2$, and $t_1 = 1 \cdot r_1$. We can apply the induction hypothesis on $q_1 \rightsquigarrow r_1$, $s_2 \rightsquigarrow t_2$, to get $t_1 \equiv t_2$ by transitivity of (Equiv). Same can be done for the right term.

The same process can be done for the (Shape) rules, but inside the `shape` construct. □

Theorem 1 (Confluence). *Given a well-typed term t if there exist t_1 and t_2 such that $t \rightsquigarrow^* t_1$ and $t \rightsquigarrow^* t_2$, then there exist t_3 and t_4 such that $t_1 \rightsquigarrow^* t_3$, $t_2 \rightsquigarrow^* t_4$ and $t_3 \equiv t_4$.*

Proof. Let us prove semi-confluence, which implies confluence. Let $t \rightsquigarrow t_1$ and $t \rightsquigarrow^* t_2$. If $t = t_2$, then result is direct; else $t \rightsquigarrow t_3 \rightsquigarrow^* t_2$. If $t_3 = t_2$, by the above result, $t_1 \equiv t_2$, which concludes. Else $t \rightsquigarrow t_3 \rightsquigarrow t_4 \rightsquigarrow^* t_2$. By the above result, $t_3 \equiv t_1$. And as $t_3 \rightsquigarrow t_4$, then $t \rightsquigarrow t_4$ by (Equiv). Therefore, $t \rightsquigarrow t_1 \rightsquigarrow t_4 \rightsquigarrow^* t_2$, and we conclude. □

C.2 Proofs for Subject Reduction

Lemma 13 (Substitution lemma). *The two following properties hold:*
- If $\Gamma, x : T'; \Delta \vdash t : T$, $\Gamma; \varnothing \vdash t' : T'$ and $\sigma = \{x \to t'\}$, then $\Gamma; \Delta \vdash \sigma(t) : T$.
- If $\Gamma; \Delta, x : T' \vdash t : T$, $\Gamma; \Delta' \vdash t' : T'$ and $\sigma = \{x \to t'\}$, then $\Gamma; \Delta, \Delta' \vdash \sigma(t) : T$.

Proof. We prove this by induction on the typing tree of t. For the first property:

- Suppose the root is (ax), meaning $\Gamma, x : T'; y : T \vdash y : T$, then one can check that we can also type directly $\Gamma; y : T \vdash y : T$, and by definition $\sigma(y) = y$ as $y \notin \mathrm{Supp}(\sigma)$, which concludes.
- Suppose the root is (ax$_c$), then either $t = y$ with $y \neq x$, and the same process can be done as above. Else, we have $\sigma(t) = \sigma(x) = t'$, which concludes by the typing of t'.
- If the rules are (ax$_0$) or (ax$_1$), then again $\sigma(t) = t$ and we can derive the typing of t with Γ instead of $\Gamma, x : T'$.
- Suppose the root is (qcase), thus $t = \texttt{qcase}\, t' \left\{|0\rangle \to t_0 \,, |1\rangle \to t_1\right\}$. By induction hypothesis, x is removed from the context of t', t_0, t_1, thus we conclude by typing $\sigma(t) = \texttt{qcase}\, \sigma(t') \left\{|0\rangle \to \sigma(t_0) \,, |1\rangle \to \sigma(t_1)\right\}$.
- Suppose the rule is (cons), meaning $\Gamma, x : T'; \Delta_i \vdash t_i : A_i$; by i.h. $\Gamma; \Delta_i \vdash \sigma(t_i) : A_i$, and thus we can type $\Gamma; \Delta \vdash c(\sigma(t_1), \ldots, \sigma(t_n)) : B$, which concludes.
- Suppose the rule is (match), the same process can be done as for (qcase), by remarking that x cannot be a bound variable of a t_i, else typing could not happen.

- Suppose the root is (abs), thus we have $\Gamma, y : T''; \Delta, x : T \vdash t : T'$; by i.h. $\Gamma; \Delta, x : T \vdash \sigma(t) : T'$, thus by (abs) we can type $\Gamma; \Delta \vdash \lambda x.\sigma(t) : T \multimap T'$, which concludes as $\lambda x.\sigma(t) = \sigma(\lambda x.t)$ as $x \notin \mathrm{Supp}(\sigma)$.
- The same process can be done for (abs$_c$): we have $\Gamma, y : T'; \Delta \vdash \lambda x.t : C \Rightarrow T$, thus we can apply the induction hypothesis to $\Gamma, x : C, y : T'; \Delta \vdash t : T$ as $x \neq y$, which gives $\Gamma, x : C; \Delta \vdash \sigma(t) : T$, and we can conclude.
- Suppose the root is (unit), then we have $\Gamma, x : T'; \varnothing \vdash t : T$, which by i.h. gives $\Gamma; \varnothing \vdash \sigma(t) : T$, and concludes as $\sigma(t)$ is also unitary.
- Suppose the root is (fix), then the same process as for (abs$_c$) can be applied.
- Suppose the root is (app), meaning we have $\Gamma, x : B^{|\rangle''}; \Delta \vdash t_1 : B^{|\rangle} \multimap B^{|\rangle'}$ and $\Gamma, x : B^{|\rangle''}; \Delta' \vdash t_2 : B^{|\rangle}$; by i.h. we have $\Gamma; \Delta \vdash \sigma(t_1) : B^{|\rangle} \multimap B^{|\rangle'}$ and $\Gamma; \Delta' \vdash \sigma(t_2) : B^{|\rangle}$, and thus by (app) we can type $\Gamma; \Delta \vdash \sigma(t_1)\sigma(t_2) : B^{|\rangle'}$ which concludes.
- The same can be done for (app$_c$) and (app$_u$).
- Suppose the root is (sup), meaning we have $\Gamma, x : T'; \Delta \vdash t_i : T$. By induction hypothesis, we get $\Gamma; \Delta \vdash \sigma(t_i) : T$, and by definition of $\perp$, $t_i \perp t_j$ implies $\sigma(t_i) \perp \sigma(t_j)$, and thus we can type by (sup) $\Gamma; \Delta \vdash \sum_{i=1}^{n} \alpha_i \cdot \sigma(t_i) : T$, which concludes by the action of σ on superpositions.
- Suppose the root of the derivation is (shape), first remark that x belongs in the Γ part, because it has no marker around it; thus we have $\Gamma, x : T'; \Delta \vdash t : B^{|\rangle}$, thus by i.h. $\Gamma; \Delta \vdash \sigma(t) : B^{|\rangle}$, and we can type by (shape) $\Gamma, [\Delta]; \varnothing \vdash \mathtt{shape}(\sigma(t)) : \mathtt{shape}(B^{|\rangle})$, which concludes.
- Suppose the root is (contr), meaning we have $\Gamma, z : T', [x : B^{|\rangle}]; \Delta, y : B^{|\rangle} \vdash t : T$. Again, as z comes with no marker and thus $z \neq x$, which means that we can treat the case as before, with a non-linear context $\Gamma, [x : B^{|\rangle}]$, and conclude.
- Suppose the root is (equiv), thus $\Gamma, x : T'; \Delta \vdash t : T$; by i.h. $\Gamma; \Delta \vdash \sigma(t) : T$. One can check that if $t \equiv t'$, then $\sigma(t) \equiv \sigma(t')$, which concludes.

A similar process can be done to prove the second property. It differs only by the fact that x is now present in only one subterm context; for the other subterm, while some x may be present by a `shape` construct, replacing it with a well-typed value of same type, one can rewrite the typing tree with the new term (possibly by remodeling the (contr) rules), and conclude. Else, we have directly $\sigma(t) = t$, thus typing is direct. □

Corollary 1. *Let $\Gamma, x_1 : T_1, \ldots, x_m : T_m; \Delta, x_{m+1} : T_{m+1}, \ldots, x_n : T_n \vdash t : T$ be a well-typed term. Then, for any substitution σ such that for all $1 \leq in$, $x_i \in \mathrm{Supp}(\sigma)$, $\Gamma; \Delta_i \vdash \sigma(x_i) : T_i$, we have $\Gamma; \Delta, \Delta_1, \ldots, \Delta_n \vdash \sigma(t) : T$.*

Proof. Direct by induction on n: the case $n = 1$ is simply the above lemma, and for $n = k + 1$, any substitution σ can be written as $\sigma = \sigma' \circ \tau$ with $\tau = \{x_1 \to \sigma(x_1)\}$; $\tau(t)$ is typable by the above lemma and removes $x_1 : A_1$ from the context, and then $\sigma'(\tau(t))$ concludes by induction hypothesis. □

Definition 7. *Let $\Gamma; \Delta \vdash t : T$ be a well-typed term. We say that t types and terminates, written t.a.t., if $t \rightsquigarrow t_1 \ldots \rightsquigarrow t_k \rightsquigarrow v$, $\Gamma; \Delta \vdash t_i : T$, $\Gamma; \Delta \vdash v : T$.*

Lemma 14. *Let $(t_i)_{1\leq i\leq n}$ be well-typed terms such that $\forall i \neq j, t_i \perp t_j$. Let $s = \sum_{i=1}^n \alpha_i \cdot t_i$, and $t = \sum_{i=1}^n \beta_i \cdot t_i$ such that s, t are t.a.t. and $\sum_{i=1}^n \alpha_i \beta_i^* = 0$. Then $s \perp t$.*

Proof. As $t_i \perp t_j$, each t_i reduces to a canonical form. Let us write them up to completion as $t_i \rightsquigarrow^* \sum_{j=1}^m \gamma_{ij} \cdot v_j$. By Lemma 11, $s \rightsquigarrow^* \sum_{j=1}^m (\sum_{i=1}^n \alpha_i \gamma_{ij}) \cdot v_j$, and $t \rightsquigarrow^* \sum_{j=1}^m (\sum_{i=1}^n \beta_i \gamma_{ij}) \cdot v_j$. By definition, this is a quasi canonical form, as some phases may be zero, but v_j are two by two distinct; proving the orthogonality condition on this will imply orthogonality for the exact canonical form. First, it is easy to see that they all have the same shape, as t_i are orthogonal, thus have the same shape. Compiling the inner product gives the following:

$$\sum_{j=1}^m \sum_{j'=1}^m (\sum_{i=1}^n \alpha_i \gamma_{ij})(\sum_{i=1}^n \beta_i \gamma_{ij'}) \delta_{v_j, v'_j} = \sum_{1\leq i,i'\leq n} \alpha_i \beta_i^* \sum_{j=1}^m \gamma_{ij} \gamma_{ij'}^* = \sum_{1\leq i,i'\leq n} \alpha_i \beta_i^* \delta_{i,i'} = 0$$

where the second equality is obtained by developing each sum, and the third by definition of $t_i \perp t_j$, and we conclude by the condition on α_i, β_i. □

Lemma 15. *Let $(t_i)_{1\leq i\leq n}$, $(t'_i)_{1\leq i\leq n}$ such that $\forall ij, t_i \perp t'_j$. Then for any α_i, β_i, $\sum_{i=1}^n \alpha_i \cdot t_i \perp \sum_{i=1}^n \beta_i \cdot t'_i$.*

Proof. Consider $(\tilde{t_i})_{1\leq i\leq 2n}$ where $\tilde{t_i} = t_i$ for $1 \leq i \leq n$, and $\tilde{t_i} = t'_i$ for $n < i \leq 2n$, and apply Lemma 14. □

Lemma 16. *Let $\Gamma; \Delta \vdash t : T$ be a pure term. Then we have the following typing tree:*

$$\cfrac{\cfrac{\cfrac{}{\Gamma_0; \Delta \vdash s : T}\ (r)}{\Gamma_1; \Delta \vdash \sigma_1(s) : T}\ (contr)}{\begin{array}{c}\vdots \\ \hline \Gamma_n; \Delta \vdash \sigma_n(s) : T\end{array}}\ (contr)$$

where $n \geq 0$, each $\Gamma_n = \Gamma$, $\sigma_n(s) = t$, r is neither (contr) nor (equiv), and each Γ_i, respectively σ_i, contains one less marked variable, contains one more mapping, as per (contr).

Proof. Let s be a pure term, and $\Gamma; \Delta \vdash t : T$ be any term. We prove by induction on typing of t that if $\theta_s(t) \neq 0$, then s satisfies the property of the lemma.

- (ax): $t = x$, $\theta_s(t) \neq 0 \implies s = x$, and we can type s directly.
- (ax$_c$), (ax$_0$) and (ax$_1$) are obtained similarly.
- (qcase): $t = \texttt{qcase}\, r \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\}$, $\theta_s(t) \neq 0 \implies s = \texttt{qcase}\, q \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\} \wedge$ $r \equiv q$ by Lemma 8 as q is pure. We can type q via (equiv) by typing r first, and then type s via (qcase).
- (cons): $t = c(t_1, \ldots, t_n)$, non-zero quantity implies $s = c(s_1, \ldots, s_n)$ with $s_i \equiv t_i$. We can type t_i by the typing of t, then s_i via (equiv), then s via (cons).

- (match): same as for (qcase).
- (abs), (abs$_c$), (rec), (unit): same as (ax).
- (app): this implies $t = t_1 t_2$, $s = s_1 s_2$, with $s_i \equiv t_i$, thus we type s_i through (equiv) and conclude.
- (app$_c$), (app$_u$): same as (app).
- (sup): $t = \sum_{i=1}^{n} \alpha_i \cdot t_i$, $\theta_s(t) = \sum_{i=1}^{n} \alpha_i \theta_s(t_i)$. As $\theta_s(t) \neq 0$, then there is $1 \leq i \leq n$ such that $\theta_s(t_i) \neq 0$. We can apply the induction hypothesis on t_i, as it has the same context and type as t.
- (shape): $t = \mathtt{shape}(r)$, $s = \mathtt{shape}(q)$ and $q \equiv r$, thus we type q through (equiv) and conclude.
- (equiv): $t \equiv t'$, $\theta_s(t) = \theta_s(t)' \neq 0$ through Lemma 7, and we apply i.h. on t' as it has the same context and type.
- (contr): $t = \sigma(r)$. We can apply the induction hypothesis on r, then reapply (contr) on the alternative tree.

Now, we can conclude as by Lemma 6, $\theta_s(s) = 1$; and any obtained tree satisfies the hypothesis. □

Lemma 17. *Let $\varnothing; x : B^{|\rangle} \vdash s : B^{|\rangle'}$, $\sigma_v = \{x \to v\}$, $\varnothing; \varnothing \vdash v : B^{|\rangle}$, $\varnothing; \varnothing \vdash v' : B^{|\rangle}$ such that $v \perp v'$ and s t.a.t. . We also suppose that any term t that terminates in less steps than s is also t.a.t. . Then $\sigma_v(s) \perp \sigma_{v'}(s)$.*

Something that we can note is that given $s \rightsquigarrow^* \alpha \cdot |0\rangle + \beta \cdot |1\rangle$, then $\mathtt{qcase}\, s \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\}$ will not always reduce to $\alpha \cdot t_0 + \beta \cdot t_1$, as $(C_\equiv)$ requires a pure term, which is not guaranteed for s. However, we will have $\mathtt{qcase}\, s \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\} \rightsquigarrow^* \alpha \cdot s_0 + \beta \cdot s_1$, where $t_i \rightsquigarrow^* s_i$. In particular, proving results of orthogonality between such terms will guarantee orthogonality between the t_i. Same goes for $\mathtt{match}$.

Proof. By induction on the typing of s:

- (ax) gives the desired result directly;
- (ax$_c$), (ax$_0$) and (ax$_1$) have an empty linear context;
- (qcase): $s = \mathtt{qcase}\, t \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\}$; as s terminates, so does t, in less steps than s, thus t reduces to a well-typed value by hypothesis. Therefore, either $\varnothing; x : B^{|\rangle} \vdash t : \text{Qbit}$; the only pure closed values of type Qbit are $|0\rangle$ and $|1\rangle$, thus $\sigma_v(t) \rightsquigarrow^* \alpha \cdot |0\rangle + \beta \cdot |1\rangle$, $\sigma_{v'}(t) \rightsquigarrow^* \gamma \cdot |0\rangle + \delta \cdot |1\rangle$, and by orthogonality $\alpha\gamma^* + \beta\delta^* = 0$. By definition, $\sigma_v(s) \rightsquigarrow^* \alpha \cdot t_0 + \beta \cdot t_1$, $\sigma_{v'}(s) \rightsquigarrow^* \gamma \cdot t_0 + \delta \cdot t_1$, which are well-typed as s is t.a.t., and we can conclude by Lemma 14. Else, $\sigma_v(t) = t$, thus $t \rightsquigarrow^* \alpha \cdot |0\rangle + \beta \cdot |1\rangle$, $\sigma_v(s) \rightsquigarrow^* \alpha \cdot \sigma_v(t_0) + \beta \cdot \sigma_v(t_1)$. In particular, $t_0 \perp t_1$, thus $\sigma_v(t_i) \perp \sigma_{v'}(t_{1-i})$, and $\sigma_v(t_i) \perp \sigma_{v'}(t_i)$ by induction hypothesis, thus we conclude by Lemma 15, as the obtained summation is t.a.t. as a reduced of s.
- (cons): there is $1 \leq i \leq n$ with $\varnothing; x : B^{|\rangle} \vdash t_i : {B^{|\rangle}}_i$. As one coordinate is orthogonal and the others are identical, orthogonality is verified for the whole term.

- (match): same as for (qcase). The only difference is that $\sigma(t)$ will reduce to a linear combination of same constructor, thus $\sigma_v(s) \rightsquigarrow^* \sum_{j=1}^m \beta_j \cdot \sigma_{w_j}(t_i)$, where $w_j \perp w'_j$, and the result is direct by induction hypothesis and by concluding with Lemma 15.
- (abs), (abs$_c$), (rec), (unit) do not yield a term of type $B^{|\rangle}$.
- (app): by typing, reduction of $t_1 t_2$ implies that t_1 reduces too to a well-typed closed value, i.e.either $\lambda x.t$, `letrec` or `unit`. Furthermore, by typing, either x is in the context of t_1, thus of t, or in the context of t_2, thus of v. In any case, by joining all substitutions, we reach a point where $s \rightsquigarrow^* \tilde{\sigma}(t)$, and we can apply the i.h. on t, as it is t.a.t. as a reduced from s.
- Same goes for (app$_c$) and (app$_u$).
- (sup): all t_i have the x in the context, thus $\sigma_v(t_i) \perp \sigma_{v'}(t'_i)$, and $\sigma_v(t_i) \perp \sigma_{v'}(t_j)$ for $i \neq j$ as $t_i \perp t_j$, thus we conclude by Lemma 15, as we apply the lemma on s, which is t.a.t. itself.
- (shape): no linear context.
- (equiv): apply the hypothesis on t, then via (Equiv) we can obtain a reduction for s.
- (contr): just an alpha-renaming of a classical variable, does not disturb the previous proofs. □

Lemma 18 (Strong Subject Reduction). *Let $\Gamma; \Delta \vdash t : T$ be a well-typed term, such that t terminates in k steps. Then t t.a.t. .*

Proof. This is proven by induction on the depth of termination of t. Let us first prove that t reduces to well-typed terms. If $k = 0$, the result is direct; suppose $k = k' + 1$, thus $t \rightsquigarrow t' \rightsquigarrow^{\leq k'} v$. We will prove that t t.a.t. by induction on the derivation of $\rightsquigarrow$. In most cases, we obtain that t' is well-typed, which implies directly that t t.a.t. by induction hypothesis on t'.

First, note that apart from (Can) and (Equiv), left terms are pure, thus we can apply Lemma 16 to gain a term with no (equiv) rule. Now, let us first suppose that there is no (contr) rule near the end of the typing tree.

- (Qcase$_0$): the only non (contr) and (equiv) rule typing this syntax is (qcase), thus $\Delta = \Delta_1, \Delta_2$ with $\Gamma; \Delta_1 \vdash |0\rangle : \text{Qbit}$ and $\Gamma; \Delta_2 \vdash t_0 : Q$. Again, as $|0\rangle$ is pure, the only no (contr) and (equiv) rule typing $|0\rangle$ is (ax$_0$). Therefore, $\Delta_1 = \emptyset$, and we can conclude by typing of t_0.
- (Qcase$_1$): same as above.
- (Match): the root of typing is (match), thus $\Delta = \Delta_1, \Delta_2$, with $\Gamma; \Delta_1 \vdash c_i(p_1^v, \dots, p_{n_i}^v) : B$ and $\Gamma, \overrightarrow{y_i} : \overrightarrow{C_i}; \Delta_2, \overrightarrow{z_i} : \overrightarrow{Q_i} \vdash t_i : B^{|\rangle}$. Again, by typing as the v_i are pure, this gives us $c :: B_1^{|\rangle}, \dots, B_n^{|\rangle} \to B$, and $\Gamma; \Delta_i^c \vdash p_i^v : B_i^{|\rangle}$, with $\Delta_1 = \Delta_1^c, \dots, \Delta_{n_i}^c$. As $\overrightarrow{x_i} = \overrightarrow{y_i}, \overrightarrow{z_i}$, we conclude using Corollary 1.
- (Lbd) : If the root is (app), then $\Delta = \Delta_1, \Delta_2$, $\Gamma; \Delta_1 \vdash \lambda x.s : T' \multimap T$ and $\Gamma; \Delta_2 \vdash v : T'$. Typing of $\lambda x.s$ implies $\Gamma; \Delta_1, x : T' \vdash s : T'$, and thus by substitution lemma on $\sigma(s)$ we conclude. The same process can be done if the root is (app$_c$).
- (Fix) and (Unit) are identical to (Lbd), up to some small tweaks.

– (Can) : we consider an alternative typing tree where all equivalence relations are at the end, as we make them commute with summation, `qcase`, `match` and constructors. Therefore, we have $t \equiv s$, where s has no (equiv) rule near the end. If s is pure, then $t = 1 \cdot s$ (by quantity), by i.h. this reduces to $1 \cdot s'$ where s' is well-typed, and using (equiv) we can type $1 \cdot s'$. Else, s is not pure. Let us first prove that given two well-typed closed terms s, t such that $s \perp t$ and that they terminate, reducing to well-typed closed value, then developing linearly preserves the orthogonality, meaning:

  - $\texttt{qcase}\, s \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\} \perp \texttt{qcase}\, t \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\}$: we have $s \rightsquigarrow^* v$ and $t \rightsquigarrow^* w$ (they terminate as we can derive orthogonality). By definition, v, w are well-typed closed values, thus $v \equiv \alpha \cdot |0\rangle + \beta \cdot |1\rangle$ and $w \equiv \gamma \cdot |0\rangle + \delta \cdot |1\rangle$. As they are orthogonal, then they will verify $\alpha\gamma^* + \beta\delta^* = 0$ (even if this is not the canonical form as some coefficients can be zero, this will still hold). Now, this indicates, by E, that $\texttt{qcase}\, s \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\} \rightsquigarrow^* \alpha \cdot s_0 + \beta \cdot s_1$ and $\texttt{qcase}\, t \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\} \rightsquigarrow^* \gamma \cdot s_0 + \delta \cdot s_1$, where $t_0 \rightsquigarrow^* s_0$ and $t_1 \rightsquigarrow^* s_1$. By Lemma 14, as $t_0 \perp t_1$, then $s_0 \perp s_1$, and thus $\alpha \cdot s_0 + \beta \cdot s_1 \perp \gamma \cdot s_0 + \delta \cdot s_1$.
  - $c(s, \ldots, t_n) \perp c(t, \ldots, t_n)$: we suppose that s, t are in first slot, other cases are done equivalently. By E, when reducing each term to a value, the first slot will contain the reduced value of s and t. Then, as they are orthogonal, and the other values are identical, they have the same shape by Lemma 10, and when writing the summation to test the zero equality, we can just develop through the first coordinate to obtain back the summation from the orthogonality of s and t.
  - $\texttt{match}\, s \{\ldots\} \perp \texttt{match}\, t \{\ldots\}$: by definition, $s \rightsquigarrow^* \sum_{i=1}^n \alpha_i \cdot w_i$ and $t \rightsquigarrow^* \sum_{i=1}^n \beta_i \cdot w_i$, by writing their canonical form, and then completing by some $0 \cdot w_i$ in order to have the same pure elements on each side. As they are orthogonal, then $\sum_{i=1}^n \alpha_i \beta_i^* = 0$. Furthermore, they must have the same shape, and be a value of type B, thus any pure value satisfies $w_i = c_k(v_i^1, \ldots, v_i^n)$. As $w_i \neq w_j$ for $i \neq j$, then one coordinate is distinct. Now, $\texttt{match}\, s \{\ldots\} \rightsquigarrow^* \sum_{i=1}^n \alpha_i \cdot s_i$ and $\texttt{match}\, t \{\ldots\} \rightsquigarrow^* \sum_{i=1}^n \beta_i \cdot t_i$, where $\sigma(w_i) \rightsquigarrow^* s_i$, and $\sigma(w_i) \rightsquigarrow^* t_i$. For any $i \neq j$, σ_i and σ_j have map one variable to a distinct pure value, as explained above; and any $v \neq w$ where v, w are pure values satisfy $v \perp w$. We can thus apply Lemma 17, and get $\sigma_i(w_i) \perp \sigma_j(w_j)$. Finally, we can conclude by Lemma 14, and as s_i, t_i are reduced terms from $\sigma_i(w_i)$, orthogonality is preserved.

  Now, as s is not pure, then there is a subterm of s that contains a (sup) typing rule, followed by some (maybe 0) linear rules, i.e. (cons), (match) or (qcase), as they are the only rules which need a pure term inside their construction to be pure. By definition of $C_\equiv$, we can develop linearly this sum to obtain $s' = \sum_{i=1}^n \alpha_i \cdot t_i$; and by construction, we can use the above result by induction, as each term is a subterm of a terminating term, thus reduces in less steps than the original term, therefore we can apply the induction hypothesis. Note that some terms may obtain a phase 0 at the end, and could have reduced in more steps; but as they will be removed, not applying this process is ok, as we will not test orthogonality with them anyway. as the base case is (sup) thus requires orthogonal terms. Therefore, by syntax, s' is a sum of orthogonal well-typed terms. We can reapply this process on each t_i until they are pure, which terminates as t_i has a smaller typing tree than s, as one (sup) rule has been removed. At the end, we obtain multiple summation of pure orthogonal values $\sum_{i=1}^{n} \alpha_i \cdot (\cdots \sum_{k=1}^{l} \beta_k \cdot t_k)$. Each sum can be typed through (sup), as the coefficients come from a (sup) rule and the above cases show that orthogonality is kept. We can then develop everything, group equivalent pure terms, and obtain the canonical form back. However, after reducing each term, we can do the reverse and rewrite the obtained term as multiple summations. Furthermore, each term has either reduced, or stayed the same as it was removed; by induction hypothesis on $t_i \rightsquigarrow^? t'_i$, t'_i is well-typed with the same context and type as t; and the orthogonality conditions are still true as we reduce terms, thus we can still type (sup) for each sum, and therefore type the big sum; as this sum is equivalent to $\sum_{i=1}^{n} \alpha_i \cdot t'_i$, by (equiv), we can type it and conclude.

– (Shape$_0$) : by typing, we have $\Gamma'; \Delta' \vdash |0\rangle : T'$ where $\Gamma = \Gamma, [\Delta'], \Delta = \emptyset$ and $T = \mathtt{shape}(T')$. Again, as $|0\rangle$ is pure, then we need to type it with (ax$_0$), which gives $\Delta' = \emptyset$ and $T = \mathtt{shape}(\mathrm{Qbit}) = \mathbb{1}$. We conclude as we can type $\Gamma; \varnothing \vdash () : \mathbb{1}$ through (cons).
– (Shape$_1$) : same as above.
– (Shape$_c$) : as the root is (shape), typing gives $\Gamma = \Gamma', [\Delta'], \Delta = \emptyset, T = [B]$ and $\Gamma; \Delta \vdash c(v_1, \ldots, v_n) : B$. Again by typing, the root being (cons), we have $\Gamma; \Delta_i \vdash v_i : B_i^{|\rangle}$ and $\Delta = \Delta_1, \ldots, \Delta_n$. Therefore, (shape) gives $\Gamma, [\Delta_i]; \varnothing \vdash \mathtt{shape}(v_i) : \mathtt{shape}(B_i^{|\rangle})$; as $[\Delta] = [\Delta_1], \ldots, [\Delta_n]$, the definition of $\tilde{c}$ concludes.
– (Shape$_s$): by Lemma 2, each v_i is well-typed with the same type as the summation, and we conclude as above.
– (Equiv): By (equiv), $\Gamma; \Delta \vdash t_1 : T$, by i.h., $\Gamma; \Delta \vdash t'_1 : T$, and by (equiv) again on $t_1 \equiv t'$, we conclude.
– (E): by i.h. on $t \rightsquigarrow t'$, $\Gamma; \Delta \vdash t' : T$, thus one can replace the typing tree of t by the typing tree of t', and any rule will still hold. This is in particular true because E is made of no superpositions nor unitary, so none of these checks happen and thus typing is preserved.

Now, given a term with some (contr) rules, we may consider the following:

– One can first remark that any (contr) rule commutes with a (qcase), (match), or any (app) rule. Therefore, for the (Qcase), (Match), (Lbd), (Fix), (Unit) rules, we can consider an alternative typing tree for t with $k \geq 0$ (contr) rules at the end, and thus have $t = \sigma_1 \ldots \sigma_k(s)$, where s has no (contr) rule near the end, thus we can apply the hypothesis above. Also note that if $t \rightsquigarrow t'$, then $t' = \sigma_1 \ldots \sigma_k(s')$ and $s \rightsquigarrow s'$. We can then apply the result above on $s \rightsquigarrow s'$, and reapply the k (contr) rules on s' to obtain the well-typedness of t'.
– The (Can) rule contains terms that all have the same context, thus any (contr) rule may actually be applied on each t_i before.

– Any (Shape) rule contains only values, thus no marked variable is actually given by the context and no (contr) rule is needed.

Therefore, the result still holds for any term. □

Lemma 4 (Subject reduction). *Let $\Gamma; \Delta \vdash t : T$ be a well-typed term, and $t \rightsquigarrow t'$. If t terminates or t is pure, then $\Gamma; \Delta \vdash t' : T$.*

Proof. If t terminates, then we can apply Lemma 18. If t is pure, one can remark that the proof in Lemma 18 for pure terms does not require termination, as (Can) is stopped from going further and needing the termination when t is pure, thus one can adapt the proof to get the result.

C.3 Proofs for Canonical Forms

Lemma 19. *Let t be a term and s be a pure term. Suppose $\theta_s(t) \neq 0$. Then t has a unique canonical form.*

Proof. Suppose $t \equiv 0 \cdot t'$. Then $\theta_s(t) = \theta_s(0 \cdot t') = 0$ by Lemma 7 and by definition of the quantity; this contradicts the initial hypothesis. We can then conclude by Lemma 9. □

Lemma 2 (Canonical form for typed terms). *Let $\Gamma; \Delta \vdash t : T$ be a well-typed term. Then t has a canonical form $\sum_{i=1}^{n} \alpha_i \cdot t_i$, and this canonical form is unique up to reordering and equivalence on the t_i. Furthermore, $\Gamma; \Delta \vdash t_i : T$.*

Proof. Let us first prove that for any well-typed t, there exists a pure term s such that $\theta_s(t) \neq 0$; we then conclude by Lemma 19. By induction on t:

– If t is pure, then take $s = t$ and result is direct;
– If the root is (cons), meaning $t = c(t_1, \dots, t_n)$, then by induction hypothesis, as t_i is typed, there is a pure term s_i such that $\theta_{s_i}(t_i) \neq 0$. We can take $s = c(s_1, \dots, s_n)$ and conclude.
– Same is done for (qcase) and (match).
– If the root is (equiv), $t \equiv t'$, we obtain the result by induction hypothesis on t' and conclude by Lemma 7.
– If the root is (contr), then $t = \sigma(t')$; we can apply the induction hypothesis on t' and take $s = \sigma(s')$ which will preserve σ.
– Finally, if the root is (sup), $t = \sum_{i=1}^{n} \alpha_i \cdot t_i$. By typing, t_i is well-typed, thus by induction hypothesis and Lemma 19, t_i has a canonical form. Furthermore, as $t_i \perp t_j$, each t_i terminates, with $t_i \rightsquigarrow^* \sum_{j=1}^{n_i} \beta_{i,j} \cdot w_j^i$, being canonical forms. One can complement each canonical form to get $t_i \rightsquigarrow^* \sum_{j=1}^{m} \beta_{i,j} w_j$, where the w_j are pure and distinct pairwise. We can thus use Lemma 11, and get that $t \rightsquigarrow^* \sum_{j=1}^{m} (\sum_{i=1}^{n} \alpha_i \beta_{i,j}) \cdot w_j$. If we suppose that $t \equiv 0 \cdot t'$, taking $t' = v$, by Theorem 1, $0 \cdot v \equiv \sum_{j=1}^{m} (\sum_{i=1}^{n} \alpha_i \beta_{i,j}) \cdot w_j$. By Lemma 7, $\theta_{w_j}(0 \cdot v) = 0 = \sum_{i=1}^{n} \alpha_i \beta_{i,j}$. One can check that by definition of $t_i \perp t_j$, the rows $(\beta_{i,1}, \dots, \beta_{i,n})$ are orthogonal, thus this would imply that $\alpha_i = 0$, which contradicts the hypothesis of (sup). Therefore, $t \not\equiv 0 \cdot t'$, by Lemma 9, t has a canonical form $\sum_{k=1}^{l} \gamma_k \cdot u_k$, and take $s = u_1$ to conclude.

Finally, one can remark that by construction of the canonical form in Lemma 9, along with an induction on typing, the pure terms obtained are well-typed; such result is possible as the canonical form is unique, thus the one that we exhibited will match.

C.4 Proofs of progress

Lemma 20. *Let t be a well-typed pure term of type $B^{|\rangle}$, and v be a value. We suppose that $t \equiv v$, then t is a value.*

Proof. As t is pure, $\theta_t(v) = \theta_t(t) = 1$. By definition of the quantity, it is easy to check that $\theta_t(v) = 1$ implies that t is a value too, thus one can conclude. □

Lemma 21. *Let t be a well-typed term of type $T_1 \rightarrowtail T_2$. Then its canonical form is $1 \cdot t'$.*

Proof. Direct by the way canonical forms are generated in Lemma 9, and by typing of the different constructs. □

Lemma 3 (Progress). *Let $\varnothing; \varnothing \vdash t : T$ be a closed term, either t is equivalent to a value, or t reduces.*

Proof. We prove this result by induction on the typing of t. First, let us consider that t is a pure term.

- The rules (ax), (ax$_c$) do not derive a closed term.
- The rules (ax$_0$), (ax$_1$) yield directly a value.
- If the root is (qcase), then $t = \texttt{qcase}\, s \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\}$, with $\varnothing; \varnothing \vdash s :$ Qbit. Either $s \rightsquigarrow s'$ and $t \rightsquigarrow \texttt{qcase}\, s' \left\{|0\rangle \to t_0\,, |1\rangle \to t_1\right\}$ through (E); or $s \equiv v$, which by Lemma 20 indicates that s is a pure value, thus $s = |i\rangle$. In any case, we can reduce through (Qcase$_i$) for $i = 0, 1$.
- Suppose t is typed with (cons), namely $\varnothing; \varnothing \vdash c(t_1, \dots, t_n) : B$. This implies $\varnothing; \varnothing \vdash t_i : B_i$, thus we can apply the induction hypothesis on t_i. If each $t_i \equiv v_i$, then $t \equiv c(v_1, \dots, v_n)$ through $C_\equiv$, which is a value. Else, we take the first t_i, starting from the right, such that $t_i \rightsquigarrow t_i'$, and then $t \rightsquigarrow c(t_1, \dots, t_i', \dots, t_n)$ by (E).
- If the root is (match), then $t = \texttt{match}\, s \,\{c_1(\overrightarrow{x_1}) \to t_1\,, \dots, c_n(\overrightarrow{x_n}) \to t_n\}$, with $\varnothing; \varnothing \vdash s : B$. Either $s \rightsquigarrow s'$ and $t \rightsquigarrow \texttt{match}\, s' \left\{\dots\right\}$ through (E); or $s \equiv v$, which by Lemma 20 indicates that s is a pure value, thus $s = \bar{c}(v_1, \dots, v_m)$. As s is of type B, then $\bar{c} \in \mathrm{Cons}(B)$, thus $\bar{c} = c_i$ and $t \rightsquigarrow t_i$ through (Match).
- The rules (abs), (abs$_c$), (rec) and (unit) yield directly a value.
- Suppose the root of the derivation is (app), meaning $t = t_1 t_2$. By typing, t_1, t_2 are also closed terms and the induction hypothesis can be applied. If $t_2 \rightsquigarrow t_2'$, then $t \rightsquigarrow t_1 t_2'$ by (E). Else, $t_2 \equiv v_2$. If $t_1 \rightsquigarrow t_1'$, then $t_1 v_2 \rightsquigarrow t_1' v_2$ by (E), and as $t \equiv t_1 v_2$, then $t \rightsquigarrow t_1' v_2$ by (Equiv). Else, $t_1 \equiv v_1$. By Lemma 21, as t_1 has a functional type, then its canonical form consists of one pure term; as v_1 has the same canonical form by unicity, this term is a value too. Thus,

$t_1 \equiv 1 \cdot v \equiv v$. Now, the only closed, well-typed values of type $C \Rightarrow T$ or $T \multimap T'$ are either $\lambda x.t'$, and as $(\lambda x.t')v_2 \rightsquigarrow \sigma(t')$ via (Lbd), then $t \rightsquigarrow \sigma(t')$ via (Equiv); or $\texttt{letrec}\, f\, x = t'$ and as $(\texttt{letrec}\, f\, x = t')v_2 \rightsquigarrow \sigma(t')$ via (Fix), then $t \rightsquigarrow \sigma(t')$ via (Equiv).

- The same proof can be done for (app_u), except that the only closed and well-typed value will be $\texttt{unit}(t')$ and we will use the reduction rule (Unit) instead of (Lbd).
- The rule (sup) does not derive a pure term.
- Suppose the root is (shape), meaning $t = \texttt{shape}(s)$, with $\varnothing; \varnothing \vdash s : B^{|\rangle}$. Note that s is actually closed because its linear and non-linear context are joined in the non-linear context of t which is empty. Then either $s \rightsquigarrow s'$ and $t \rightsquigarrow \texttt{shape}(s')$ via (E); or $s \equiv v$. Either $v = |i\rangle$, and t reduces through (Shape_i) and (Equiv); or $v = c(v_1, \ldots, v_n)$ and t reduces through (Shape_c) and (Equiv); or v is a summation, as it is well-typed it has a canonical form w, and by transitivity $t \equiv w$, thus t reduces through (Shape_s) and (Equiv). Any other value cannot be typed with a type $B^{|\rangle}$.
- Suppose the root of the derivation is (equiv), namely we have $t' \equiv t$. By hypothesis, t is closed and thus t' is too. Therefore, either $t' \equiv v$ and by transitivity of $\equiv$, $t \equiv v$; else $t' \rightsquigarrow t''$ and then $t \rightsquigarrow t''$ by (Equiv).
- The rule (contr) does not derive a closed term.
- Finally, the rule (contr) cannot yield a closed term.

Now, let t be a well-typed closed term; let us consider its canonical form $t' = \sum_{i=1}^{n} \alpha_i \cdot t_i$. If t' is a value, then t is equivalent to a value, and we conclude. Else, any term t_i verifies $t_i \rightsquigarrow^? t'_i$ for a given t'_i, by definition, and thus $t' \rightsquigarrow \sum_{i=1}^{n} \alpha_i \cdot t'_i$ by (Can), and thus $t \rightsquigarrow \sum_{i=1}^{n} \alpha_i \cdot t'_i$ by (Equiv), which concludes. □

C.5 Orthogonality proofs

Theorem 2 (Undecidability of orthogonality). *Deciding orthogonality between two well-typed terms is Π_2^0-complete.*

Proof. Let us first prove the Π_2^0-hardness. Let us recall the definition of Programming Computable Functions (PCF), which is defined with the following types and terms grammar:

$$A, B ::= \texttt{nat} \mid A \to B$$
$$t, t_1, t_2 ::= x \mid \lambda x.t \mid t_1 t_2 \mid \texttt{fix}\ t \mid \texttt{n} \mid \texttt{succ} \mid \texttt{pred} \mid \texttt{ifz}\ t_1 t_2$$

Each term has a corresponding typing rule, and thus PCF is a typed language. It is easy to see that our language can encode PCF: any term is already in our language, apart from pred and ifz, which can be encoded as follows:

$$\texttt{pred} = \lambda x.\texttt{match}\, x \left\{0 \to 0, S\ n \to n\right\}$$
$$\texttt{ifz} = \lambda x.\lambda y.\lambda n.\texttt{match}\, n \left\{0 \to x, S\ m \to y\right\}$$

Therefore, any term of PCF can be encoded into a term in Hyrql through a total computable function. The Universal Halt problem on PCF is known to be Π^0_2-hard. We define `UHalt` as the set of terms of PCF, thus terms in our language, terminating over any input, meaning $tt_1 \dots t_n$ terminates.

Given a well-typed term t from PCF of type $T_1 \rightarrowtail \dots \rightarrowtail T_n \rightarrowtail B^{|\rangle}$ (such type can always be obtained by the Lemma 5), we define

$$g(t) = (tx_1 \dots x_n, |0\rangle), (tx_1 \dots x_n, |1\rangle),$$

where $x_1, \dots, x_n$ are neither free nor bound variables of t. If $t \in \texttt{UHalt}$, then it terminates over any input. In particular, this implies that $tx_1 \dots x_n$ terminates for any substitution, reducing to a value v; and as $(v, |0\rangle) \perp (v, |1\rangle)$, $g(t) \in \texttt{ORTHO}$. If $g(t) \in \texttt{ORTHO}$, then t must terminate for any substitution of $x_1, \dots, x_n$, thus terminates for any value. Therefore, $t \in \texttt{UHalt}$. Thus, $t \in \texttt{UHalt} \iff g(t) \in \texttt{ORTHO}$, $\texttt{UHalt} \leq \texttt{ORTHO}$, and as `UHalt` is Π^0_2-hard, it concludes.

The fact that $\texttt{ORTHO} \in \Pi^0_2$ is direct by definition. Given t, t' be two terms of the same type with $\mathrm{FV}(t) = \mathrm{FV}(t') = \{x_1, \dots, x_n\}$, and let $\sigma_{v_1,\dots,v_n} = \{x_i \to v_i\}_{1 \leq i \leq n}$, then orthogonality decision can be written up as:

$$\forall v_1, \dots \forall v_n, \exists k \in \mathbb{N},\ \sigma_{v_1,\dots,v_n}(\texttt{shape}(t)) \rightsquigarrow^{\leq k} v_s, \sigma_{v_1,\dots,v_n}(\texttt{shape}(t')) \rightsquigarrow^{\leq k} v'_s$$
$$\sigma(t) \rightsquigarrow^* \sum_{i=1}^{n} \alpha_i \cdot v_i \in \texttt{CAN}, \sigma(t') \rightsquigarrow^* \sum_{j=1}^{m} \beta_j \cdot w_j \in \texttt{CAN}$$
$$v_s = v'_s \wedge \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j^* \delta_{v_i, w_j} = 0$$

The inside property is decidable: checking if a term reducing in less than k steps to a variable, written $t \rightsquigarrow^{\leq k} v$, is decidable; checking the first equality is decidable as there is no superposition (computed in $\mathcal{O}(n)$ with n the size of v), and the equality to 0 of the double sum and equality is decidable as we have restricted the scalars in $\bar{\mathbb{Q}}$. Thus the inside property is Π^0_0; by definition, `ORTHO` belongs in Π^0_2, and thus is Π^0_2-complete. □

Proposition 1. *Given terms s, t that are well-typed, of finite type, closed, and terminating, it is decidable whether they are orthogonal. More, if s, t terminate in polynomial time, then orthogonality can be computed in polynomial time.*

Proof. One can check that any value of type T is of size $|v| \leq |T|_d$, by definition of the depth, therefore we use n as a bound on the size of any value of such type. In order to compute and decide `ORTHO`, we first need to reduce s and t to values, which is done in f, g steps. Then, computing $\texttt{shape}(s)$ and $\texttt{shape}(t)$ is done linearly in the size of the value, same for computing equality, thus this is done in $\mathcal{O}(n)$ steps. For the computation of the equality, one needs to compute the canonical forms for the reduced values of s and t. In particular, we can complement each canonical form, so that testing δ_{v_i, w_j} is done directly by comparing the index. This implies checking all terms of the canonical form, and comparing it with all the other terms, thus checking $\mathcal{O}(n)$, comparing it with $\mathcal{O}(n)$ terms, and comparison is done in the worst case in $\mathcal{O}(n)$ steps. Then, we compute each

$\alpha_i\beta_j^*$, and then we have to sum n terms and checking nullity. As we have assumed to reduce the set of complex phases such that this can be done polynomially, we assume it is not the operation with the most cost. Such calculus needs to be done in the worst case n^3 times. □

C.6 Proofs for linearity and isometries

Lemma 22. *Let $\Gamma; \Delta, x : T \vdash s : B^{|\rangle}$; suppose that it terminates for any substitution $\{x \to v\}$. Let $v = \sum_{i=1}^n \alpha_i \cdot v_i$ a well-typed value of type T. Let us write $\sigma = \{x \to v\}$, and $\sigma_i = \{x \to v_i\}$. Then $\sigma(s)$ and $\sum_{i=1}^n \alpha_i \cdot \sigma_i(s)$ reduce to the same value.*

Proof. By induction on the typing of s. Note that if x is not in the linear context of a given t, then $\sigma(t) = \sigma_i(t)$; substitution would only work inside a `shape` construct, and by definition v and v_i will reduce to the same shape.

- (ax) is direct, the other axiom rules do not verify the hypothesis.
- (qcase): $s = \texttt{qcase}\, t \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\}$. By typing, either x is present in the context of t, thus we can use the linearity of `qcase`; or it is in t_0 and t_1, $s \rightsquigarrow^* \alpha \cdot \tilde{t_0} + \beta \cdot \tilde{t_1}$, where $t_i \rightsquigarrow^* \tilde{t_i}$, and we conclude by i.h. on each t_i. Note that the use of $\tilde{t_i}$ is required because the left part of the (E) reduction rule requires a pure term, thus each part may not reduce at the same time.
- (cons): $s = c(t_1, \ldots, t_n)$. By typing, x is present in one t_i, thus we can conclude by linearity of c.
- (match) goes as for (qcase).
- Any (abs) rule does not have a good type.
- Given $s = t_1 t_2$, in any case we will reach a point where $s \rightsquigarrow^* \tau(t')$, where t' is typed by weak subject reduction, and $x \in \mathrm{Supp}(\tau)$, thus we can conclude.
- (sup): $s = \sum_{j=1}^m \beta_j \cdot t_j$: we can use the induction hypothesis on each t_j, use $\equiv$ to permute terms and conclude. □

Proposition 2 (Linear functional terms). *Let $\varnothing; \varnothing \vdash t : T_1 \multimap T_2$ such that $t\, w$ terminates for any input $\varnothing; \varnothing \vdash w : T_1$. For any $\varnothing; \varnothing \vdash \sum_{i=1}^n \alpha_i \cdot v_i : T_1$, there exist t_1, t_2 such that $t\,(\sum_{i=1}^n \alpha_i \cdot v_i) \rightsquigarrow^* t_1$ and $\sum_{i=1}^n \alpha_i \cdot t\, v_i \rightsquigarrow^* t_2$, and $t_1 \equiv t_2$.*

Proof. Direct by Lemma 22. □

Lemma 23. *The following hold for s, t, s_i being well-typed closed terminating terms:*

- *If $s \perp t$, then $\langle s, t\rangle = 0$.*
- *$\langle s, t\rangle = \overline{\langle t, s\rangle}$;*
- *$\langle \sum_{i=1}^n \alpha_i \cdot s_i, t\rangle = \sum_{i=1}^n \alpha_i \cdot \langle s_i, t\rangle$;*
- *$\langle s, s\rangle = 1$.*

Proof. The first two properties are direct by definition of the inner product. The third property can also be obtained directly from Lemma 11: given $s_i \rightsquigarrow^* \sum_{j=1}^m \beta_{ij} \cdot w_j$, $t \rightsquigarrow^* \sum_{k=1}^l \gamma_k \cdot u_k$:

$$\langle \sum_{i=1}^n \alpha_i \cdot s_i, t\rangle = \sum_{\substack{1\leq j\leq m\\1\leq k\leq l}} (\sum_{i=1}^n \alpha_i\beta_{ij})\gamma_k^*\delta_{w_j,u_k} = \sum_{i=1}^n \alpha_i \cdot \sum_{\substack{1\leq j\leq m\\1\leq k\leq l}} \beta_{ij}\gamma_k^*\delta_{w_j,u_k} = \sum_{i=1}^n \alpha_i\langle s_i, t\rangle$$

Finally, the last result can be obtained by induction on the length of termination, by case analysis on the root of the typing tree of s. Among the interesting cases:

- For (qcase), by induction hypothesis, $t \rightsquigarrow^* \alpha\cdot|0\rangle + \beta\cdot|1\rangle$, with $|\alpha|^2+|\beta|^2 = 1$. We will get $s \rightsquigarrow \alpha \cdot s_0 + \beta \cdot s_1$, where $t_i \rightsquigarrow^* s_i$, thus they are orthogonal; we conclude by induction hypothesis on the s_i, and by the three first properties.
- For (match), $t \rightsquigarrow^* \sum_{i=1}^n \alpha_i \cdot v_i$, where the v_i are orthogonal as they are distinct; by Lemma 17, $\sigma_i(t_j)$ will be pairwise orthogonal, thus we can conclude as for (qcase).
- For (sup), let $t_i \rightsquigarrow^* \sum_{j=1}^m \beta_{ij} \cdot v_j$. Then, by Lemma 11 :

$$\begin{aligned}\langle s, s\rangle &= \sum_{j=1}^m (\sum_{i=1}^n \alpha_i\beta_{ij})(\sum_{i'=1}^n \alpha_{i'}\beta_{i'j})^*\delta_{v_j,v'_j}\\ &= \sum_{1\leq i,i'\leq n} \alpha_i\alpha_i^* \sum_{j=1}^m \beta_{ij}\beta_{i'j} = \sum_{1\leq i,i'\leq n} \alpha_i\alpha_i^*\delta_{i,i'} = 1\end{aligned}$$

  where the last inequality is a consequence of $t_i \perp t_{i'}$ for $i \neq i'$, and the induction hypothesis for $i = i'$ (as by Lemma 18, the reduced of t_i is well-typed).
- For (equiv), if $s \equiv t$, and $t \rightsquigarrow^* v$, then so does s (up to equivalence), thus we can conclude. □

Proposition 3 (Quantum linear terms are isometries). *Let $\Gamma; \Delta \vdash t : Q_1 \multimap Q_2$; if it terminates for any input, then t is an isometry.*

Proof. Let v, w be well-typed closed values, and let their canonical form, up to completion $\sum_{i=1}^n \alpha_i \cdot v_i$ and $\sum_{i=1}^n \beta_i \cdot w_i$. By Proposition 2, $\sigma(t)v$ reduces to the same value as $\sum_{i=1}^n \alpha_i \cdot \sigma(v)$, same for $\sigma(t)w$, therefore:

$$\begin{aligned}\langle \sigma(t)v, \sigma(t)w\rangle &= \langle \sum_{i=1}^n \alpha_i \cdot \sigma(t)v_i, \sum_{i=1}^n \beta_i \cdot \sigma(t)w_i\rangle = \sum_{1\leq i,i'\leq n} \alpha_i\beta_i^*\langle \sigma(t)v_i, \sigma(t)v'_i\rangle\\ &= \sum_{1\leq i,i'\leq n} \alpha_i\beta_i^*\delta_{i,i'} = \langle v, w\rangle\end{aligned}$$

where, the first equality comes from confluence, the second from linearity of the inner product, and the third by the fact that each term is of norm 1 for $i = i'$, and when $i \neq i'$, $v_i \perp v'_i$ as they are pure, and we can use Lemma 17 and Lemma 23 to conclude. □

D A translation algorithm from Hyrql to STTRS

Before providing our algorithm, we define formally the reduction of a STTRS, by introducing ane equivalence relation, similarly to `Hyrql`. equivalence relation, similarly to `Hyrql`. This equivalence relation $\equiv_{\mathcal{R}}$ is defined by the rules in Table 7, where $\tilde{\mathcal{F}} = \mathcal{F} \setminus \{\mathtt{shape}, \mathtt{unit}\}$.

$$\mathtt{t}_1 + \mathtt{t}_2 \equiv_{\mathcal{R}} \mathtt{t}_2 + \mathtt{t}_1 \qquad \mathtt{t}_1 + (\mathtt{t}_2 + \mathtt{t}_3) \equiv_{\mathcal{R}} (\mathtt{t}_1 + \mathtt{t}_2) + \mathtt{t}_3 \qquad 1 \cdot \mathtt{t} \equiv_{\mathcal{R}} \mathtt{t} \qquad \mathtt{t} + 0 \cdot \mathtt{t}' \equiv_{\mathcal{R}} \mathtt{t}$$

$$\alpha \cdot (\beta \cdot \mathtt{t}) \equiv_{\mathcal{R}} \alpha\beta \cdot \mathtt{t} \qquad \alpha \cdot (\mathtt{t}_1 + \mathtt{t}_2) \equiv_{\mathcal{R}} \alpha \cdot \mathtt{t}_1 + \alpha \cdot \mathtt{t}_2 \qquad \alpha \cdot t + \beta \cdot t \equiv_{\mathcal{R}} (\alpha + \beta) \cdot t$$

$$\mathtt{t}(\overrightarrow{\mathtt{t}_1}, \textstyle\sum_{i=1}^n \alpha_i \cdot \mathtt{s}_i, \overrightarrow{\mathtt{t}_2}) \equiv_{\mathcal{R}} \sum_{i=1}^n \alpha_i \cdot t(\overrightarrow{\mathtt{t}_1}, \mathtt{s}_i, \overrightarrow{\mathtt{t}_2}), \quad \mathtt{t} \in \tilde{\mathcal{F}}$$

Table 7. Equivalence relation on terms of a STTRS

This allows us to define how terms of a STTRS reduce.

Definition 8 (Rewrite relation). *We define the TRS-context as a term with one hole, namely given by the following grammar:*

$$\mathtt{C} ::= \diamond \mid \mathtt{t}(\overrightarrow{\mathtt{t}}, \mathtt{C}, \overrightarrow{\mathtt{v}}) \mid \mathtt{C}(\overrightarrow{\mathtt{v}})$$

In this language, we restrict substitutions to maps from variables to patterns. We write $s \to_{\mathcal{R}} t$, if the following conditions are respected:

- $\mathtt{s} \equiv_{\mathcal{R}} \sum_{i=1}^n \alpha_i \cdot \mathtt{s}_i$ *and* $\mathtt{t} \equiv_{\mathcal{R}} \sum_{i=1}^n \alpha_i \cdot \mathtt{t}_i$, *where* $\alpha_i \neq 0$ *and* $\mathtt{s}_i$ *are pure terms;*
- *For any* $1 \leq i \leq n$, *either* $\mathtt{s}_i = \mathtt{t}_i = \mathtt{v}$, *or* $\mathtt{s}_i = \mathtt{C}_i[\sigma_i l_i]$ *and* $\mathtt{t}_i = \mathtt{C}_i[\sigma_i r_i]$ *with* $l_i \to r_i \in \mathcal{R}$.
- *There is at least one* $\mathtt{s}_i$ *that is not a value.*

We now provide below an algorithm to convert any term of the language into a STTRS. While we could provide an algorithm that converts any term, reduction in both worlds will not yield a semantically equal term. Therefore, we first provide a syntactic restriction to our language.

Definition 9 (Admissible term). *We define the admissible syntax by the following grammars:*

$$t^a ::= x \mid |0\rangle \mid |1\rangle \mid c(t_1^a, \ldots, t_n^a) \mid \sum_{i=1}^{n} \alpha_i \cdot t_i^a \mid \mathtt{shape}(t^a) \mid t^f$$

$$t^f ::= x \mid (\lambda x.t^s)^c \mid (\mathtt{letrec}\, f\, x = t^s)^c \mid \mathtt{unit}(t^a) \mid t^f t^a$$

$$t^s ::= t^a \mid \mathtt{qcase}\, x \left\{|0\rangle \to t_0^s, |1\rangle \to t_1^s\right\} \mid \mathtt{match}\, x \left\{c_1(\overrightarrow{x_1}) \to t_1, \ldots, c_n(\overrightarrow{x_n}) \to t_n\right\} \mid \lambda x.t^s$$

The notation $(t)^c$ *states that* $\mathrm{FV}(t) = \emptyset$. *Finally, we say that* t *is an admissible term, if it is well-typed, closed, its syntax is given by the grammar of* t^a, *and no variable* x *appears in more than one pattern-matching construct.*

The key points of this restriction are the following:

– Any pattern-matching construct is applied only to a variable.
– Higher-orders is restricted to variables, closed values, or applications. Furthermore, any recursive construct is the last abstraction before obtaining a closed term. Such restriction allows to easily transform the higher order term into a function symbol.

Such restriction allows to decompose the reduction in multiple steps, by introducing multiple function symbols, and will be necessary later to obtain a semantic equivalence between both worlds. Furthermore, this does not discard any program which could be written in `Hyrql`, as shown below:

Proposition 5 (Admissible transformation). *Let* $\varnothing; \varnothing \vdash t : T$ *be a well-typed terminating term of size* $|t| = n$. *Suppose* $T = T_1 \rightarrowtail \ldots \rightarrowtail T_n \rightarrowtail B^{|\rangle}$. *Then there exists an admissible term* s *such that for any* $\varnothing; \varnothing \vdash v_i : T_i$, $tv_1 \ldots v_n \rightsquigarrow^{\leq k} v$, $sv_1 \ldots v_n \rightsquigarrow^{\leq l} v$, *and* $l = \mathcal{O}(kn^2)$. *Furthermore, this transformation is done in* $\mathcal{O}(n^2)$ *steps.*

Proof. First, let us show that given $\Gamma; \Delta \vdash t : T$, we have $\Gamma; \Delta \vdash s : T$, with no variable x happening in two pattern-matching constructs, and s satisfying the admissible grammar, such that for any substitution σ, $\sigma(t)v_1 \ldots v_n$ and $\sigma(s)v_1 \ldots v_n$ are joinable.

By induction on the syntax of t:

– $x, |0\rangle, |1\rangle$ already satisfies the conditions.
– $t = \texttt{qcase}\, t' \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\}$. We can apply the induction on t', t_0, and t_1 to get s', s_0, s_1. Let x be a variable that is not already used. Taking $x_1, \ldots, x_n = \mathrm{FV}(s') \cup \mathrm{FV}(s_0) \cup \mathrm{FV}(s_1)$, let us write $s = (\lambda x.\lambda x_1.\ldots\lambda x_n.\texttt{qcase}\, x \left\{|0\rangle \to s_0\, , |1\rangle \to s_1\right\})s'x_1 \ldots x_n$. One can check that as s', s_0, s_1 are well-typed by induction hypothesis, then so is s, with the good type and context. Furthermore, s is derived from the grammar of t^a. By hypothesis, we also have $s' \rightsquigarrow^* v$ and $t' \rightsquigarrow^* v$, $t \rightsquigarrow^* \texttt{qcase}\, v \left\{|0\rangle \to t_0\, , |1\rangle \to t_1\right\}$, and $s \rightsquigarrow^* \texttt{qcase}\, v \left\{|0\rangle \to s_0\, , |1\rangle \to s_1\right\}$; we can conclude, as s_i, t_i terminate and are joinable.
– $t = c(t_1, \ldots, t_n)$: by induction on each t_i, we have s_i with the same context such that they are joinable. Thus, one can write $s = c(s_1, \ldots, s_n)$, which is well-typed with the corresponding type and context, is derived from the admissible grammar, and as $T = B$, by $C_{\equiv}$, any s_i terminates, thus $s \rightsquigarrow^* c(v_1, \ldots, v_n)$ and $t \rightsquigarrow^* c(v_1, \ldots, v_n)$, which concludes. By definition, as s_i satisfies the admissible syntax, any variable in a pattern-matching construct is captured, thus we can also satisfy this condition via α-renaming.
– $t = \texttt{match}\, t' \left\{\ldots\right\}$: we can do the same process as for `qcase`.

– $t = \lambda x.t'$: by induction on t', we have s' such that they are joinable; let $s = \lambda x.s'$. Suppose $T = T_1 \rightarrowtail \ldots \rightarrowtail T_n \rightarrowtail B^{|\rangle}$. Then, $sv_1 \ldots v_n \rightsquigarrow \sigma(s')v_2 \ldots v_n$, and $tv_1 \ldots v_n \rightsquigarrow \sigma(t')v_2 \ldots v_n$.

Now, when t is closed, then s is closed too, thus s is an admissible term. Note that the translation goes linearly through the term, and adds $\mathcal{O}(n)$ abstractions, in the worst case, to fit the syntax. Therefore, the translation has a cost of $\mathcal{O}(n^2)$. The same process goes for the bound on the terminations: for each step, we add at most $\mathcal{O}(n)$ abstractions, therefore the scaling of the reduction is $\mathcal{O}(kn^2)$. As t terminates, then so does s, and joinable implies that they reduce to the same value; and as t is closed, $\sigma(t) = t$ and $\sigma(s) = s$, and we recover the shape of the theorem. $\square$

Note that this proof is constructive. The construction will be used in Algorithm 3.

We now present an algorithm that aims to translate any admissible term into a term-rewriting system. The main part of the algorithm is defined in Algorithm 1, that builds the STTRS inductively on the syntax of the term. This algorithm applied to a term s returns three sets C, R, S which are called the partial rules of s, where:

– C contains the current rules, i.e. rules of the shape $l \to r, \sigma$, indicating that under a substitution σ, s has a rewrite rule $l \to r$;
– R contains the rewrite rules that are already well-formed, in particular rewrite rules for function symbols that were introduced when parsing s;
– S is a set $\{t \to \mathtt{f}\}$ indicating the function symbols $\mathtt{f}$ introduced as well as the original term t in the base language.

Intuitively, each rule will correspond to a specific choice of branches in the pattern-matching constructs. The entry point of this algorithm is done in Algorithm 2, which takes an admissible term and will call Algorithm 1 from it. This algorithm only returns all the needed rewrite rules R and the set of introduced symbols S.

We also introduce in Table 8 an interpretation, that maps any term t of the language to a term $M = \langle t \rangle_{\mathcal{S}}$ of the produced STTRS. Note that this depends on the set of function symbols, as we need it to interprete any term t in S as its corresponding function symbol. We also introduce an interpreation $\langle - \rangle_{\mathcal{T}}$ of the type of `Hyrql` to the types of a STTRS. Given two substitutions σ, δ, we denote $\sigma \circ \delta$ as the ordered composition of substitutions, meaning $(\sigma \circ \delta)(t) = \sigma(\delta(t))$.

We now give the properties that our translation verifies. First, as the system is build inductively on the syntax, such computation is done in polynomial time.

Proposition 6 (Translation complexity). *Let t be an admissible term of size k. Then the computation* `TranslateAdmissible`(t) *is done in* $\mathcal{O}(k^2)$ *steps.*

Furthemore, our algorithm also produces well-defined systems for admissible terms.

Algorithm 1: Main translation algorithm

```
Function Translate($s$):
    Data: An admissible subterm $s$
    Result: The partial rewrite rules for $s$
    switch $s$ do
        when $x$ or $|0\rangle$ or $|1\rangle$: return $\{\bot \to s, \epsilon\}, \emptyset, \emptyset$
        when qcase $x \left\{|0\rangle \to t_0 \,, |1\rangle \to t_1\right\}$ : begin
            $C_i, R_i, S_i \longleftarrow$ Translate($t_i$), $i = 0, 1$
            for $i$ in $\{0, 1\}$, $(l \to r, \sigma)$ in $C_i$ do
                $\sigma_x \longleftarrow \{x \to |i\rangle\}$ and $\sigma \longleftarrow \sigma \circ \sigma_x$ and $r \longleftarrow \sigma_x(r)$
            return $C_0 \cup C_1, R_0 \cup R_1, S_0 \cup S_1$
        when $c(t_1, \ldots, t_n)$: begin
            $C_i, R_i, S_i \longleftarrow$ Translate($t_i$), $1 \le i \le n$
            assert $C_i = \{\bot \to r_i, \epsilon\}, 1 \le i \le n$
            return $\{\bot \to c(r_1, \ldots, r_n), \epsilon\}, \cup_{i=1}^n R_i, \cup_{i=1}^n S_i$
        when match $x \{c_1(\overrightarrow{x_1}) \to t_1 \,, \ldots, c_n(\overrightarrow{x_n}) \to t_n\}$ : begin
            $C_i, R_i, S_i \longleftarrow$ Translate($t_i$), $1 \le i \le n$
            $C \longleftarrow \emptyset$
            for $i$ in $\{1, \ldots, n\}$, $(l \to r, \sigma)$ in $C_i$ do
                assert $\sigma = \sigma_b \circ \sigma_x = \sigma_x \circ \sigma_b$ and $\sigma(c(\overrightarrow{x_i})) = \sigma_x(c(\overrightarrow{x_i})) = v$
                $\sigma_c \longleftarrow \{x \to v\}$
                $C \longleftarrow C \cup \{l \to \sigma_c(r), \sigma_b \circ \sigma_c\}$
            return $C', \cup_{i=1}^n R_i, \cup_{i=1}^n S_i$
        when $\lambda x.t$ or letrec $f\, x = t$: begin
            $C, R, S \longleftarrow$ Translate($t$)
            foreach $(l \to r, \sigma) \in C$ do  $l \longleftarrow \sigma(x) l$
            if $\mathrm{FV}(t) = \varnothing$ then
                $\mathtt{f}$ = FctSymbol()
                if $t =$ letrec $f\, x = t$ then $\tau \longleftarrow \{f \to \mathtt{f}\}$ else $\tau \longleftarrow \epsilon$
                $R \longleftarrow R \cup \{\mathtt{f}\, l \to \tau(r) \mid (l \to r, \sigma) \in C\}$
                $C \longleftarrow \{\bot \to \mathtt{f}, \epsilon\}$ and $S \longleftarrow S \cup \{s \to \mathtt{f}\}$
            return $C, R, S$
        when unit($t$): begin
            $C, R, S \longleftarrow$ Translate($t$)
            if $t \to \mathtt{f} \in S$ then
                $S \longleftarrow S \cup \{s \to \mathtt{f}\} \setminus \{t \to \mathtt{f}\}$
                $r \longleftarrow \sigma(r)$ for $\sigma = \{\mathtt{f} \to \mathtt{unit}(\mathtt{f})\}$
            else
                foreach $(l \to r, \sigma) \in C$ do  $r \longleftarrow \mathtt{unit}(r)$
            return $C, R, S$
        when $t_1 t_2$: begin
            $C_i, R_i, S_i \longleftarrow$ Translate($t_i$), $i = 1, 2$
            assert $C_i = \{\bot \to r_i, \epsilon\}, i = 1, 2$
            return $\{\bot \to r_1 r_2, \epsilon\}, R_1 \cup R_2, S_1 \cup S_2$
        when $\sum_{i=1}^n \alpha_i \cdot t_i$: begin
            $C_i, R_i, S_i \longleftarrow$ Translate($t_i$), $1 \le i \le n$
            assert $C_i = \{\bot \to r_i, \epsilon\}, 1 \le i \le n$
            return $\{\bot \to \sum_{i=1}^n \alpha_i \cdot r_i, \epsilon\}, \cup_{i=1}^n R_i, \cup_{i=1}^n S_i$
        when shape($t$): begin
            $C, R, S \longleftarrow$ Translate($t$)
            foreach $l \to r, \sigma \in C$ do  $r \longleftarrow \mathtt{shape}(r)$
            return $C, R, S$
```

$$\langle x\rangle_{\mathcal{S}} = x \qquad \langle c(t_1,\dots,t_n)\rangle_{\mathcal{S}} = c\langle t_1\rangle_{\mathcal{S}}\dots\langle t_n\rangle_{\mathcal{S}} \qquad \langle \textstyle\sum_{i=1}^n \alpha_i\cdot t_i\rangle_{\mathcal{S}} = \sum_{i=1}^n \alpha_i\cdot\langle t_i\rangle_{\mathcal{S}}$$

$$\langle t_1 t_2\rangle_{\mathcal{S}} = \langle t_1\rangle_{\mathcal{S}}\langle t_2\rangle_{\mathcal{S}} \qquad \langle t\rangle_{\mathcal{S}} = \mathtt{f} \text{ if } t\to\mathtt{f}\in S$$

$$\langle \mathtt{shape}(t)\rangle_{\mathcal{S}} = \mathtt{shape}\langle t\rangle_{\mathcal{S}} \qquad \langle \mathtt{unit}(t)\rangle_{\mathcal{S}} = \mathtt{unit}\langle t\rangle_{\mathcal{S}}$$

$$\langle B^{|\rangle}\rangle_{\mathcal{T}} = B^{|\rangle} \qquad \langle T\rightarrowtail T'\rangle_{\mathcal{T}} = \langle T\rangle_{\mathcal{T}}\to\langle T'\rangle_{\mathcal{T}}$$

Table 8. Interpretation from a term of the language into a term of a STTRS, with a set of terms interpretations S

Algorithm 2: Translation algorithm of admissible terms

```
Function TranslateAdmissible(s):
    Data: An admissible term $s$
    Result: The rewrites rules for $s$ and all the interpretations of the
            introduced function symbols
    $C, R, S \longleftarrow$ Translate(s)
    assert $C = \{\perp \to r, \epsilon\}$
    if $r = st$ then $\mathtt{f} \longleftarrow$ FctSymbol(), $R \longleftarrow R \cup \{\mathtt{f} \to r\}$, $S \longleftarrow S \cup \{t \to \mathtt{f}\}$
    $R \longleftarrow R \cup \{\mathtt{unit}\, x\, y \to x\, y\}$
    $R \longleftarrow R \cup \{\mathtt{shape}\ |0\rangle \to (), \mathtt{shape}\ |1\rangle \to ()\}$
    $R \longleftarrow R \cup \{\mathtt{shape}(\alpha\cdot|0\rangle + \beta\cdot|1\rangle) \to ()\}$
    $R \longleftarrow R \cup \{\mathtt{shape}(c(x_1,\dots,x_n)) \to \tilde{c}(\mathtt{shape}(x_1),\dots,\mathtt{shape}(x_n)) \mid B \in$
     $\mathbb{B}, c \in \mathrm{Cons}(B)\}$
    return $R, S$
```

Theorem 3 (Well-definedness). *Let s be an admissible term, and $R, S =$ `TranslateAdmissible(s)`. Then R is a well-defined STTRS.*

Lemma 24. *Let s be an admissible subterm of type $B^{|\rangle}$, and $C, R, S =$ `Translate(s)`. Then for any $(l \to r, \sigma) \in C$, $l = \perp$.*

Proof. Direct by induction on s. □

Definition 10. *Let σ be a substitution. We say that σ is a pattern substitution if for any $x \in \mathrm{Supp}(\sigma)$, $\sigma(x)$ is a pattern. Given two pattern substitutions σ, σ',*

Algorithm 3: Translation algorithm for any term

```
Function TranslateEntry(t):
    Data: A term $t$
    Result: The rewrites rules for $s$ and all the interpretations of the
            introduced function symbols
    $s \longleftarrow$ Use proof of Proposition 5 on $t$
    $R, S \longleftarrow$ TranslateAdmissible(s)
    return $R, S$
```

with $\mathrm{Supp}(\sigma) \cup \mathrm{Supp}(\sigma') = x_1, \dots, x_n$*, we say that they are non-overlapping if* $\sigma(x_1) \dots \sigma(x_n)$ *and* $\sigma'(x_1) \dots \sigma'(x_n)$ *cannot overlap.*

Lemma 25. *Let s be an admissible subterm of type T, and $C, R, S = \texttt{Translate}(s)$. Then one of the two following is true:*

(P1) $\forall (l \to r, \sigma) \in C$*,* $l = \perp$*, and r is of type* $\langle T \rangle_{\mathcal{T}}$*;*
(P2) $\exists k \geq 1, T_1, \dots, T_k, T', \forall (l \to r, \sigma) \in C$*,* $l = p_1 \dots p_k$*, where p_i is pattern of type* $\langle T_i \rangle_{\mathcal{T}}$*, r is of type* $\langle T' \rangle_{\mathcal{T}}$*, every variable happens at most once in l, and* $T = T_1 \rightarrowtail \dots \rightarrowtail T_k \rightarrowtail T'$*;*

Furthermore, the set $\{\sigma \mid (l \to r, \sigma) \in C\}$ *is a set of pattern substitutions that are two by two non-overlapping.*

Proof. This can be proven directly by induction on s:

– $s = x, |0\rangle, |1\rangle$ is direct.
– $s = \texttt{qcase}\, x \left\{ |0\rangle \to t_0 \,, |1\rangle \to t_1 \right\}$: by typing, s is of type Q, thus $B^{|\rangle}$. Therefore, by Lemma 24, any left part of a rule in $C_0 \cup C_1$ is $\perp$, and (P1) is verified. If σ is a pattern substitution, then so is $\sigma \circ \{x \to |i\rangle\}$; and given two substitutions, either they come from a different C_i thus $\sigma(x)$ will prevent any overlap; or they come from the same, and thus we conclude by induction hypothesis.
– $s = c(t_1, \dots, t_n)$: by induction hypothesis, as each t_i is of type $B^{|\rangle}$, any r_i has the same type as t_i, thus $c(\sigma(r_1), \dots, \sigma(r_n))$ has the same type as s, and any left term is $\perp$, thus (P1) is verified.
– $s = \texttt{match}\, x \{c_1(\overrightarrow{x_1}) \to t_1 \,, \dots, c_n(\overrightarrow{x_n}) \to t_n\}$: by typing, as t_i is of type $B^{|\rangle}$, any rule in C_i has a left term $l = \perp$. Furthermore, one can check that $\sigma_c(r)$ has the same type as r, and as s has the same type as each t_i, (P1) is verified. The fact that the two substitutions are non-overlapping either comes from the fact that the σ_b are non-overlapping, or if the σ_x are non-overlapping, as they are in the construction of σ_c, the result will still hold.
– $s = \lambda x.t, \texttt{letrec}\, f\, x = t$: by definition, $\sigma(x)$ is a pattern. Therefore, either (P1) was verified for t, and then $l = \sigma(x)$, or (P2) was verified and then $\sigma(x) p_1 \dots p_k$ still satisfies (P2). The typing condition is direct as s has type $T_1 \rightarrowtail T_2$. Substitutions are untouched thus the result is still good. If we enter the if condition, then $\{\perp \to \mathtt{f}, \epsilon\}$ satisfies directly the conditions.
– $s = \texttt{unit}(t)$: by syntax, t is either a variable or a closed function, thus the rules of t are such that $l = \perp$, thus we can conclude.
– $s = t_1 t_2, \sum_{i=1}^{n} \alpha_i \cdot t_i$ is proven in the same fashion as $c(t_1, \dots, t_n)$.
– $s = \texttt{shape}(t)$: as t is of type $B^{|\rangle}$, left part of a rule is $\perp$, thus we can conclude.

Proof (Proof of Theorem 3). Proving this lemma means that the STTRS obtained by the algorithm satisfies Definition 5. Let us prove them one by one. Note that orthogonality of rules is equivalent to non-overlapping and left-linear rules, which is what is proven here.

Given any rule $l \to r \in R$, it comes from two spots. Either it was created in the call of $\texttt{Translate}(t)$, therefore $t = \lambda x.s$ (same reasoning can be done if

$t = \mathtt{letrec}\, f\, x = s$). Now, this implies that $l = \mathtt{f}l'$ and that $(l' \to r, \sigma) \in C$ satisfied the property (P2) from Lemma 25. This first gives us that $l = \mathtt{f}p_1 \dots p_k$, which is the wanted shape. Furthermore, as $\mathtt{f}$ is of the same type as t, we can use the typing results of (P2) to conclude that l, r have the same type. Any variable also happens at most once in l by (P2). Furthermore, one can check that $\mathrm{FV}(l) \supseteq \mathrm{FV}(r)$. Indeed, suppose there is $x \in \mathrm{FV}(r)$ and $x \notin \mathrm{FV}(l)$. By definition of the algorithm, if x cannot by captured by a `match` construct, else it would have been susbtituted and would no more be in r. If it is captured in $\lambda x.s'$ or $\mathtt{letrec}\, f\, x = s'$, then we would have $x \in \mathrm{FV}(l)$. Therefore, the only possibility is if $x \in \mathrm{FV}(t)$, which cannot be true as t is closed.

The other possibility is if the rewrite rule was created in $\mathtt{TranslateAdmissible}(t)$. The rewrite rules for `shape` and `unit` satisfy directly the hypothesis; and if the rule is $\mathtt{f} \to r$, with $\perp \to r \in C$, it satisfies (P1), and thus the result is obtained directly.

The non-overlapping property also comes from Lemma 25, as any symbol $\mathtt{f}$ is associated with a unique term, thus generated for all rules satisfying this property. Each symbol also has the same arity, again by the same lemma. □

Finally, we prove that the runtime-complexity between `Hyrql` and our STTRS is related. To do so, we first prove the following lemma, stating that one rewrite rule preserves the semantics. The fact that we need to use another set than $\mathcal{S}$ is explained in Remark 3.

Lemma 26 (Single rewrite rule semantics). *Let* $\mathcal{R}, \mathcal{S} = \mathtt{TranslateAdmissible}(s)$ *for a given admissible term s. Let $l \to r \in \mathcal{R}$. Then, for any well-typed closed term t, any substitution σ and any set $\mathcal{S}'$, such that $\langle t \rangle_{\mathcal{S}'} = \sigma l$, $t \rightsquigarrow^{\leq |s|} t'$ where $\langle t \rangle_{\mathcal{S}'} = \sigma r$.*

Remark 2. Note that Lemma 26 is only defined for a rewrite rule, and not for any reduction $\to_{\mathcal{R}}$. This is because of our reduction strategy: superpositions are reduced in parallel, and each term is not guaranteed to reduce in the same time. Given $s = (\lambda x.x)$, and $s \to \mathtt{id} \in S$, one can write $t = \frac{1}{\sqrt{2}} \cdot (s\,|0\rangle, s\,|0\rangle) + \frac{1}{\sqrt{2}} \cdot (s(s\,|0\rangle), |1\rangle)$; and while any term of the superposition satisfies Lemma 26, t does not satisfy it. However, semantics is preserved at the end; and this does not disturb the length of reduction, as when all terms reach the end of their reduction, it is the case for the main term.

Lemma 27. *Let* $C, R, S = \mathtt{Translate}(t)$ *for a well-typed term t. Suppose that $t = t^a$ is given by the admissible grammar. Then $C = \{\perp \to r, \epsilon\}$ with $\langle t \rangle_{\mathcal{S}} = r$.*

Proof. By induction on the syntax of t, by considering all the possible cases in both grammar.

- $t = x$: $C = \{\perp \to x, \epsilon\}$, thus the result is direct. Same goes for $|0\rangle$ and $|1\rangle$.
- $t = c(t_1^a, \dots, t_n^a)$: by induction hypothesis on t_i^a, there is only one rule, thus $C = \{\perp \to c(r_1, \dots, r_n), \epsilon\}$; and $\langle t \rangle_{\mathcal{S}} = c(\langle t_1 \rangle_{\mathcal{S}}, \dots, \langle t_n \rangle_{\mathcal{S}}) = c(r_1, \dots, r_n) = r$.

- $t = (\lambda x.t^s)^c$: as t is closed, then $C = \{\perp \to \mathtt{f}, \epsilon\}$, thus we conclude. Same goes for $t = (\mathtt{letrec}\, f\, x = t^s)^c$.
- $t = \mathtt{unit}(t^a)$: by induction, t^a has one rule, and $\langle t\rangle_{\mathcal{S}} = \mathtt{unit}(\langle t^a\rangle_{\mathcal{S}})$, which concludes.
- $t = t^f t^a$: direct by concatenation of both rules.
- $t = \sum_{i=1}^{n} \alpha_i \cdot t_i^a$: same as for $c(t_1, \ldots, t_n)$.
- $t = \mathtt{shape}(t^a)$: same as for $\mathtt{unit}(t)$.
- $t = t^f$: direct by induction hypothesis. □

Lemma 28. *Let $C, R, S =$ `Translate(`s`)` where s is a well-typed term of size k. Then for any $l \to r, \sigma \in C$, and for any substitution δ:*

(R1) Either $l = \perp$, and $\delta(\sigma(s)) \rightsquigarrow^{\leq k} r'$ where $\langle r'\rangle_{\mathcal{S}} = \delta(r)$;
(R2) or $l = p_1 \ldots p_k$, and $\delta(\sigma(s)p_1 \ldots p_k) \rightsquigarrow^{\leq k} r'$ where $\langle r'\rangle_{\mathcal{S}} = \delta(r)$.

Proof. By induction on s. Note that any admissible term satisfies directly (R1) by Lemma 27, therefore we only discuss the other cases.

- Suppose $s = \mathtt{qcase}\, x\, \{|0\rangle \to t_0\, , |1\rangle \to t_1\}$. Let $l \to r, \sigma \in C$. By definition of the algorithm, $\sigma = \sigma' \circ \sigma_x$, with $\sigma_x = \{x \to |i\rangle\}$, $r = \sigma_x(r')$ and $l \to r', \sigma' \in C_i$. Furthermore, by Lemma 24, $l = \perp$. Therefore, given any δ, $\delta(\sigma(s)) = \mathtt{qcase}\, |i\rangle\, \{|0\rangle \to \delta(\sigma(t_0))\, , |1\rangle \to \delta(\sigma(t_1))\} \rightsquigarrow \delta(\sigma(t_i))$. Then, by commutation, $\delta(\sigma(t)) = \delta(\sigma_x(\sigma'(t))) \rightsquigarrow^{\leq |t|} \delta(\sigma_x(r')) = \delta(r)$. By definition, $|s| \geq |t_i| + 1$, thus $\delta(\sigma(s)) \rightsquigarrow^{\leq |s|} \delta(r)$, which concludes.
- Suppose $s = \mathtt{match}\, t\, \{\ldots\}$; by syntax, $t = x$. Now, any rule in C is of the shape $(l \to \sigma_x(r), \sigma_b \circ \sigma_c)$, where $l \to r, \sigma \in C_i$ for a given i. Let any substitution δ, and let $\sigma' = \sigma_b \circ \sigma_c$.

$$\delta(\sigma'(s)) = \mathtt{match}\, c_i(\delta(\sigma_x(x_i^1)), \ldots, \delta(\sigma_x(x_i^{n_i})))\, \{c_i(\overrightarrow{x_i}) \to \delta(\sigma'(t_i))\} \rightsquigarrow \tau(\delta(\sigma'(t_i))),$$

  with $\tau = \{x_i^j \to \delta(\sigma_x(x_i^j))\}$. One can check that $\tau \circ \delta = \delta \circ \sigma_x$, and $\sigma_x \circ \sigma' = \sigma_c \circ \sigma$ by commuting the substitutions. This implies that $\tau(\delta(\sigma'(t_i))) = \delta(\sigma_c(\sigma(t))) \rightsquigarrow^{\leq |t_i|} \delta(\sigma_c(r))$, which is the expected result. Furthermore, $|s| \geq |t_i| + 1$, thus $\delta(\sigma'(s)) \rightsquigarrow^{\leq |s|} \delta(\sigma_x(r))$, which concludes.
- Suppose $s = \lambda x.t$. If s has no free variables, then the result is direct as $C = \{\perp \to \mathtt{f}, \epsilon\}$. Else, for any obtained rule, we have $l = \sigma(x)l'$ with $l' \to r, \sigma$ a rule of t. Therefore, $\sigma(s)\sigma(x)l' = \lambda x.\sigma(t)\sigma(x)l' \rightsquigarrow \delta(\sigma(t))l' = \sigma(t)l' \rightsquigarrow^{\leq |t|} r$; where $\delta = \{x \to \sigma(x)\}$ that actually satisfies $\delta \circ \sigma = \sigma$. As $|s| = |t| + 1$, then the bound on reduction is satisfied. The same can be done for $\mathtt{letrec}\, f\, x = t$. □

Proof (Proof of Lemma 26). Let $l \to r \in R$; such rule can come from two places. Either $t = \lambda x.s, \mathtt{letrec}\, f\, x = s$, thus it comes from Algorithm 1. In this case, $l = fl'$, where $l' \to r, \sigma$ satisfies (R2) from Lemma 28. As t is closed, $\sigma(t) = t$, thus the result is direct. Else, the rule has been made in Algorithm 2. Either it is a `shape`or `unit`rule, which gives directly the result by definition; or $l = \mathtt{f}$, and $\perp \to r \in C$, with C coming from `Translate(`t`)`. By definition, (R1) is verified, therefore we also obtain the result directly, as t is closed. □

We can now prove the main theorem.

Theorem 4 (Reduction complexity relation). *Let $\varnothing;\varnothing \vdash s_i : T_i$ be admissible terms for $1 \leq i \leq n$. Let $\mathcal{R}_i, \mathcal{S}_i = \texttt{TranslateAdmissible}(s_i)$. Let $\mathcal{R} = \cup_{1\leq i \leq n}\mathcal{R}_i$, $\mathcal{S} = \cup_{1\leq i\leq n}\mathcal{S}_i$, and $m = \max_{1\leq i\leq n}|s_i|$. Let t be a closed term of type $B^{|\rangle}$, and let $L = \langle t\rangle_{\mathcal{S}}$. If L reduces to a value M in k steps, then $t \rightsquigarrow^{\leq mk} v$, and $\langle v\rangle_{\mathcal{S}} = M$.*

Lemma 29. *Let $\varnothing;\varnothing \vdash sv_1 \dots v_n : T$ be an admissible term. Suppose that $v_k = \sum_{j=1}^{n} \gamma_j \cdot w_j$. Then, $sv_1 \dots v_n$ and $\sum_{j=1}^{n} \gamma_j \cdot sv_1 \dots w_j \dots v_n$ are joinable.*

Proof. This is proven by induction on the syntax of s. The admissible syntax imposes that $s = \lambda x_1. \dots \lambda x_n.s'$ (or `letrec`, but this has the same behaviour); and as we can type v_k, then $T_k = Q_k$. Therefore, x_k is used once in s'. and by construction, either x_k is used in a pattern-matching construct, on which we have linearity, either in a constructor, on which we also have linearity, or in an application, on which we can use the induction hypothesis to conclude. □

Proof (Proof of Theorem 4). We prove this by induction on k. The case $k = 0$ being direct, suppose $k = k' + 1$, therefore $L \to_{\mathcal{R}} N$, and N reduces to M in k' steps.

First, suppose that L is pure. This implies that we have a context $\mathtt{C}$, a substitution σ and a rule $l \to r \in R$ such that $L = \mathtt{C}(\sigma l)$ and $N = \mathtt{C}(\sigma r)$. The fact that $L = \langle t\rangle_{\mathcal{S}}$ implies that $t = \mathtt{C}(s)$ with $\langle s\rangle_{\mathcal{S}} = \sigma l$. As $l \to r$ comes from one R_i, we can apply Lemma 26 and get that $s \rightsquigarrow^{\leq |s_i|} s'$ where $\langle s'\rangle_{\mathcal{S}} = \sigma r$. Furthermore, one can remark that as L is pure, so is s, and so will be any element of the chain of reduction from s to s'. As any $\mathtt{C}$ can be written as a E, and $|s_i| \leq m$, we can chain m (E) reductions, to get that $t \rightsquigarrow^{\leq m} \mathtt{C}(s')$; and $\langle \mathtt{C}(s')\rangle_{\mathcal{S}} = N$. We can then use the induction hypothesis on N, L, and conclude that $t \rightsquigarrow^{\leq m} \mathtt{C}(s') \rightsquigarrow^{\leq k'm} v$, thus $t \rightsquigarrow^{\leq km} v$ with $\langle v\rangle_{\mathcal{S}} = M$.

Now, let L be a general term, that may not be pure. Let us first express it as $L \equiv_{\mathcal{R}} \sum_{i=1}^{n} \alpha_i \cdot L_i$, where we do not use the $\equiv_{\mathcal{R}}$ rule for linearity over a function symbol. This process can be done in the same fashion for t, meaning $t \equiv \sum_{i=1}^{n} \alpha_i \cdot t_i$, where $\langle t_i\rangle_{\mathcal{S}} = L_i$. Now, we write each L_i under its canonical form $L_i \equiv_{\mathcal{R}} \sum_{j=1}^{l} \beta_{ij} L_{ij}$, where L_{ij} is pure. As we have use all the other $\equiv_{\mathcal{R}}$ rules, this implies that $t_i = s_i v_i^1 \dots v_i^n$, $t_i' = \sum_{j=1}^{l} \beta_{ij} \cdot t_{ij}$ with $t_{ij} = s_i v_{ij}^1 \dots v_{ij}^n$, where we have developed multiple times a coordinate v_i^k from t_i. One can check that by multiple uses of Lemma 29, t_i and t_i' are joinable. This also implies that t and $\sum_{i=1}^{n} \alpha_i \cdot \sum_{j=1}^{m} \beta_{ij} \cdot t_{ij}$ are joinable, with $\langle t_{ij}\rangle_{\mathcal{S}} = L_{ij}$. Now, as L_{ij} is pure, we have L_{ij} reducing to M_{ij} in k steps, and $t_{ij} \rightsquigarrow^{\leq mk} v_{ij}$, with $\langle v_{ij}\rangle_{\mathcal{S}} = M_{ij}$. By multiple calls to Lemma 11, $\sum_{i=1}^{n} \alpha_i \cdot \sum_{j=1}^{m} \beta_{ij} \cdot t_{ij} \rightsquigarrow^{\leq mk} \sum_{i=1}^{n} \alpha_i \cdot \sum_{j=1}^{m} \beta_{ij} \cdot v_{ij}$. And as t is joinable with the left part, $t \rightsquigarrow^{\leq mk} v$ where $v = \sum_{i=1}^{n} \alpha_i \cdot \sum_{j=1}^{m} \beta_{ij} \cdot v_{ij}$. One can prove a result similar to Lemma 11 to also get that $\sum_{i=1}^{n} \alpha_i \cdot \sum_{j=1}^{m} \beta_{ij} \cdot L_{ij}$, thus L by $\equiv_{\mathcal{R}}$ reduces to $\sum_{i=1}^{n} \alpha_i \cdot \sum_{j=1}^{m} \beta_{ij} \cdot N_{ij}$ is k steps; by definition, $N \equiv_{\mathcal{R}} \sum_{i=1}^{n} \alpha_i \cdot \sum_{j=1}^{m} \beta_{ij} \cdot N_{ij}$, thus $\langle v\rangle_{\mathcal{S}} = N$, and we can conclude. □

This result can be expressed in an easier way in the following special case:

Corollary 2. *Let* $\varnothing;\varnothing \vdash s : B_1^{|\rangle} \rightarrowtail \ldots \rightarrowtail B_n^{|\rangle} \rightarrowtail B^{|\rangle}$*, and let* $\mathcal{R}, \mathcal{S} =$ `Translate`(s)*. Let any* $\varnothing;\varnothing \vdash v_i : B_i^{|\rangle}$*, and let* $L = \langle s v_1 \ldots v_n \rangle_{\mathcal{S}}$*. If* L *reduces to a value* M *in* k *steps, then* $s v_1 \ldots v_n \rightsquigarrow^{\leq k|s|} v$ *where* $\langle v \rangle_{\mathcal{S}} = M$*.*

Remark 3. Both results need to consider more than one set of rules and function symbols. Indeed, coming back on the example of `map`, in order to give an interpretation of `map` $\phi\, x$, when ϕ is closed, we need to provide a function symbol for ϕ, thus to run the algorithm on ϕ.

Proposition 4. *For any well-typed* `Hyrql` *term* t*,* $\mathcal{R} =$ `TranslateEntry`(t) *is a well-defined STTRS. Furthermore, if* $\mathcal{R}$ *terminates on any input* s *in time* $f(|s|)$*, then* $t\, s$ *reduces in* $\mathcal{O}(|s|^3\, f(|s|))$ *and* $\Omega(f(|s|))$ *steps.*

Proof. This is a combination of Theorem 3, Proposition 5, and Theorem 4.