A programming language characterizing quantum polynomial time

Emmanuel Hainry, Romain Péchoux, Mário Silva

A programming language characterizing quantum polynomial time

Emmanuel Hainry, Romain Péchoux, Mário Silva

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

Abstract. We introduce a first-order quantum programming language, named FOQ, whose terminating programs are reversible. We restrict FOQ to a strict and tractable subset, named PFOQ, of terminating programs with bounded width, that provides a first programming language-based characterization of the quantum complexity class FBQP. Finally, we present a tractable semantics-preserving algorithm compiling a PFOQ program to a quantum circuit of size polynomial in the number of input qubits.

1 Introduction

Motivations. Quantum computing is an emerging and promising computational model that has been in the scientific limelight for several decades. This phenomenon is mainly based on the advantage of quantum computers over their classical competitors, based on the use of purely quantum properties such as superposition and entanglement. The most notable example being Shor's algorithm for finding the prime factors of an integer [16], which is almost exponentially faster than the most efficient known classical factoring algorithm and which is expected to have spin-offs in cryptography (RSA encryption, ...).

Whether due to the fragility of quantum systems, namely the engineering problem of maintaining a large number of qubits in a coherent state, or by lack of reliable technological alternatives, quantum computing is typically described at a level close to hardware. One can think non exhaustively, to quantum circuits [9,12], to measurement-based quantum computers [5,7] or to circuit description languages [14]. This low-level machinery restricts drastically the abstraction and programming ease offered by these models and quantum programs currently suffer from the comparison with their classical competitors, which have many high-level tools and formalisms based on more than 50 years of scientific research, engineering development, and practical and industrial applications.

In order to solve these issues, a major effort is made to realize the promise of a quantum computer, which requires the development of different layers of hardware and software, together referred to as the *quantum stack*. Our paper is part of this line of research. We focus on the highest layers of the quantum stack: quantum programming languages and quantum algorithms. We seek to better understand what can be done efficiently on a quantum computer and we are particularly interested in the development of quantum programming languages where program complexity can be certified automatically by some static analysis technique.

Contribution. Towards this end, we take the notion of polynomial time computation as our main object of study. Our contributions are the following.

- We introduce a quantum programming language, named FOQ, that includes first-order recursive procedures. The input of a FOQ program consist in a sorted set of qubits, a list of pairwise distinct qubit indexes. A FOQ program can apply to each of its qubits basic operators corresponding to unary unitary operators. The considered set of operators has been chosen in accordance with [17] to form a universal set of gates.
- After showing that terminating FOQ programs are reversible (Theorem 1), we restrict programs to a strict subset, named PFOQ, for *polynomial time* FOQ. The restrictions put on a PFOQ programs are tractable (*i.e.*, can be decided in polynomial time, see Theorem 2), ensure that programs terminate on any input (Lemma 1), and prevent programs from having any exponential blow up (Lemma 2).
- We show that the class of functions computed by PFOQ programs is *sound* and *complete* for the quantum complexity class FBQP. FBQP is the functional extension of *bounded-error quantum polynomial time*, known as BQP [3], the class of decision problems solvable by a quantum computer in polynomial time with an error probability of at most $\frac{1}{3}$ for all instances. Hence the language PFOQ is, to our knowledge, the first programming language characterizing quantum polynomial time functions. Soundness (Theorem 3) is proved by showing that any PFOQ program can be simulated by a quantum Turing machine running in polynomial time [3]. The completeness of our characterization (Theorem 6) is demonstrated by showing that PFOQ programs strictly encompass Yamakami's function algebra, known to be FBQP-complete [17].
- We also describe an algorithm, named **compile** (based on the subroutines described in Algorithms 1 and 2), that compiles any PFOQ program to a quantum circuit acting on n qubits and of size polynomial in n, for all n. The existence of such circuits is not surprising, as a direct consequence of Yao's characterization of the class BQP in terms of uniform families of circuits of polynomial size [18]. However, a constructive generation based on Yao's algorithm is not satisfactory because of the use of quantum Turing machines which makes the circuits complex and not optimal (in size). We show that, in our setting, circuits can be effectively computed and that the **compile** algorithm is tractable (Theorem 7).

Our programming language FOQ and the restriction to PFOQ are illustrated throughout the paper, using the Quantum Fourier Transform QFT as a leading algorithm (Example 1).

Related work. This paper belongs to a long standing line of works trying to specify, understand, and analyze the semantics of quantum programming languages, starting with the cornerstone work of Selinger [15]. The motivations in restricting the considered programs to PFOQ were inspired by the works on *implicit computational complexity*, that seek to characterize complexity classes

by putting restrictions (type systems or others) on standard programming languages and paradigms [1,10,13]. These restrictions have to be implicit (*i.e.*, not provided by the programmer) and tractable. Among all these works, we are aware of two results [17] and [6] studying polynomial time computations on quantum programming languages, works from which our paper was greatly inspired. [6] provides a characterization of BQP based on a quantum lambda-calculus. Our work is an extension to FBQP with a restriction to first-order procedures. Last but not least, [6] is based on Yao's simulation of quantum Turing machines [18] while we provide an explicit algorithm for generating circuits of polynomial size. Our work is also inspired by the function algebra of [17], that characterizes FBQP: our completeness proof shows that any function in [17] can be simulated by a PFOQ program (Theorem 6). However, we claim that FOQ is a more general language for FBQP in so far that it is much less constraining (in terms of expressive power) than the function algebra of [17]: any function of [17] can be, by design, transformed into a PFOQ program, whereas the converse is not true.

2 First-order quantum programming language

Syntax and well-formedness. We consider a quantum programming language, called FOQ for First-Order Quantum programming language, that includes basic data types such as Integers, Booleans, Qubits, Operators, and Sorted Sets of qubits, lists of finite length where all elements are different. A FOQ program has the ability to call first-order (recursive) procedures taking a sorted set of qubits as a parameter. Its syntax is provided in Figure 1.

Let x denote an integer variable and $\bar{\mathrm{p}}$, $\bar{\mathrm{q}}$ denote sorted sets variables. The size of the sorted set stored in $\bar{\mathrm{q}}$ will be denoted by $|\bar{\mathrm{q}}|$. We can refer to the i-th qubit in $\bar{\mathrm{q}}$ as $\bar{\mathrm{q}}[i]$, with $1 \leq i \leq |\bar{\mathrm{q}}|$. Hence, each non-empty sorted set variable $\bar{\mathrm{q}}$ can be viewed as a list $[\bar{\mathrm{q}}[1], \ldots, \bar{\mathrm{q}}[|\bar{\mathrm{q}}|]]$. The empty sorted set, of size 0, will be denoted by nil and $\bar{\mathrm{q}} \ominus [i]$ will denote the sorted set obtained by removing the qubit of index i in $\bar{\mathrm{q}}$. For notational convenience, we extend this notation by $\bar{\mathrm{q}} \ominus [i_1, \ldots, i_k]$, for the list obtained by removing the qubits of indexes $i_1, \ldots, i_k$ in the sorted set $\bar{\mathrm{q}}$.

The language also includes some constructs U^f to represent (unary) unitary operators, for some total function $f \in \mathbb{Z} \to [0, 2\pi) \cap \tilde{\mathbb{R}}$. The function f is required to be polynomial-time approximable: its output is restricted to $\tilde{\mathbb{R}}$, the set of real numbers that can be approximated by a Turing machine for any precision 2^{-k} in time polynomial in k.

A FOQ *program* $\mathrm{P}(\bar{\mathrm{q}})$ consists of a sequence of *procedure declarations* D followed by a *program statement* S, ε denoting the empty sequence. In what follows, we will sometimes refer to program $\mathrm{P}(\bar{\mathrm{q}})$ simply as P. Let $var(\mathrm{S})$ be the set of variables appearing in the statement S. Let $|\mathrm{P}|$ be the size of program P, that is the total number of symbols in P.

A procedure declaration **decl** proc[x]($\bar{\mathrm{p}}$){S} takes a sorted set parameter $\bar{\mathrm{p}}$ and some optional integer parameter x as inputs. S is called the *procedure*

(Integers)	i	$\triangleq$	$n \mid \mathrm{x} \mid \mathrm{i} + n \mid \mathrm{i} - n \mid \lvert\mathrm{s}\rvert$, with $n \in \mathbb{N}$
(Booleans)	b	$\triangleq$	$\mathrm{i} > \mathrm{i} \mid \mathrm{i} \geq \mathrm{i} \mid \mathrm{i} = \mathrm{i} \mid \mathrm{b} \wedge \mathrm{b} \mid \mathrm{b} \vee \mathrm{b} \mid \neg \mathrm{b}$
(Sorted Sets)	s	$\triangleq$	$\mathrm{nil} \mid \bar{\mathrm{q}} \mid \mathrm{s} \ominus [\mathrm{i}]$
(Qubits)	q	$\triangleq$	$\mathrm{s}[\mathrm{i}]$
(Operators)	$\mathrm{U}^f(\mathrm{i})$	$\triangleq$	$\mathrm{NOT} \mid \mathrm{R}_Y^f(\mathrm{i}) \mid \mathrm{Ph}^f(\mathrm{i})$, with $f \in \mathbb{Z} \to [0, 2\pi) \cap \tilde{\mathbb{R}}$
(Statements)	S	$\triangleq$	**skip**; $\mid$ q ***=** $\mathrm{U}^f(\mathrm{i})$; $\mid$ S S $\mid$ **if** b **then** S **else** S $\mid$ **qcase** q **of** $\{0 \to \mathrm{S}, 1 \to \mathrm{S}\}$ $\mid$ **call** proc[i](s);
(Procedure declarations)	D	$\triangleq$	$\varepsilon \mid$ **decl** proc[x]($\bar{\mathrm{p}}$){S}, D
(Programs)	$\mathrm{P}(\bar{\mathrm{q}})$	$\triangleq$	$\mathrm{D} :: \mathrm{S}$

Fig. 1: Syntax of FOQ programs

statement, proc is the *procedure name* and belongs to a countable set Procedures. We will write $\mathrm{S}^{\mathrm{proc}}$ to refer to S and proc $\in$ P holds if proc is declared in D.

Statements include a no-op instruction, applications of a unitary operator to a qubit (q ***=** $\mathrm{U}^f(\mathrm{i})$;), sequences, (classical) conditionals, *quantum cases*, and *procedure calls* (**call** proc[i](s);). A quantum case **qcase** q **of** $\{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}$ provides a quantum control feature that will execute statements S_0 and S_1 in superposition. For example, the $CNOT$ gate on qubits $\bar{\mathrm{q}}[i]$ and $\bar{\mathrm{q}}[j]$, for $i, j \in \mathbb{N}$, $i \neq j$, can be simulated by the following statement:

$$\mathrm{CNOT}(\bar{\mathrm{q}}[i], \bar{\mathrm{q}}[j]) \triangleq \textbf{qcase } \bar{\mathrm{q}}[i] \textbf{ of } \{0 \to \textbf{ skip};, 1 \to \bar{\mathrm{q}}[j] \textbf{ *=} \mathrm{NOT};\}.$$

Throughout the paper, we restrict our study to *well-formed* programs, that is, programs $\mathrm{P} = \mathrm{D} :: \mathrm{S}$ satisfying the following properties: $var(\mathrm{S}) \subseteq \{\bar{\mathrm{q}}\}$; $\forall \mathrm{proc} \in \mathrm{P}$, $var(\mathrm{S}^{\mathrm{proc}}) \subseteq \{\mathrm{x}, \bar{\mathrm{p}}\}$; procedure names declared in D pairwise are distinct; for each procedure call, the procedure name is declared in D.

Semantics. Let $\mathcal{H}_{2^n}$ be the *Hilbert space* $\mathbb{C}^{2^n}$ of n qubits. We use Dirac notation to denote a quantum state $|\psi\rangle \in \mathcal{H}_{2^n}$. Each $|\psi\rangle \in \mathcal{H}_{2^n}$ can be written as a superposition of bitstrings of size n: $|\psi\rangle = \sum_{w \in \{0,1\}^n} \alpha_w |w\rangle$, with $\alpha_w \in \mathbb{C}$ and $\sum_w |\alpha_w|^2 = 1$. The *length* $\ell(|\psi\rangle)$ of the state $|\psi\rangle$ is n. Given two matrices M, N, we denote by $M^\dagger$ the transpose conjugate of M and by $M \otimes N$ the tensor product of M by N. $\langle\psi|$ is equal to $|\psi\rangle^\dagger$ and $|\psi\rangle\langle\phi|$ and $\langle\psi|\phi\rangle$ are respectively the inner product and outer product of $|\psi\rangle$ and $|\phi\rangle$. Let I_n be the identity matrix in $\mathbb{C}^{n \times n}$. Given $m \leq n$ and $i \in \{0, 1\}$, define $|i\rangle_m \triangleq I_{2^{m-1}} \otimes |i\rangle \otimes I_{2^{n-m}}$ and $\langle i|_m \triangleq (|i\rangle_m)^\dagger$.

A function $[\![\mathrm{U}^f]\!] \in \mathbb{Z} \to \tilde{\mathbb{C}}^{2\times 2}$ is associated to each U^f as follows:

$$[\![\mathrm{NOT}]\!](n) \triangleq \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ [\![\mathrm{R}_Y^f]\!](n) \triangleq \begin{pmatrix} \cos(f(n)) & -\sin(f(n)) \\ \sin(f(n)) & \cos(f(n)) \end{pmatrix},\ [\![\mathrm{Ph}^f]\!](n) \triangleq \begin{pmatrix} 1 & 0 \\ 0 & e^{if(n)} \end{pmatrix},$$

where $\tilde{\mathbb{C}}$ is the set of complex numbers whose both real and imaginary parts are in $\tilde{\mathbb{R}}$. One can check easily that each matrix $M \triangleq [\![\mathrm{U}^f]\!](n) \in \tilde{\mathbb{C}}^{2\times 2}$ is unitary, *i.e.*, it satisfies $M^\dagger M = M M^\dagger = I_2$.

Let $\mathbb{B}$ to be the set of Boolean values $b \in \{\textbf{false}, \textbf{true}\}$. For a given set X, let $\mathcal{L}(X)$ be the set of lists of elements in X. Let $l = [x_1, \ldots, x_m]$, with

$x_1, \ldots, x_m \in X$, denote a list of m-elements in $\mathcal{L}(X)$ and $[\,]$ be the empty list (when $m = 0$). For $l, l' \in \mathcal{L}(X)$, $l@l'$ denotes the concatenation of l and l'. $hd(l)$ and $tl(l)$ represent the tail and the head of l, respectively. Lists of integers will be used to represent Sorted Sets. They contain pointers to qubits (*i.e.*, indexes) in the global memory.

We interpret each basic data type τ as follows: $[\![\text{Integers}]\!] \triangleq \mathbb{Z}$, $[\![\text{Booleans}]\!] \triangleq \mathbb{B}$, $[\![\text{SortedSets}]\!] \triangleq \mathcal{L}(\mathbb{N})$, $[\![\text{Qubits}]\!] \triangleq \mathbb{N}$, and $[\![\text{Operators}]\!] \triangleq \tilde{\mathbb{C}}^{2\times 2}$. Each basic operation $\text{op} \in \{+, -, >, \geq, =, \wedge, \vee, \neg\}$ of arity n, with $1 \leq n \leq 2$, has a type signature $\tau_1 \times \ldots \times \tau_n \to \tau$ fixed by the program syntax. For example, the operation + has signature Integers × Integers → Integers. A total function $[\![\text{op}]\!] \in [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!] \to [\![\tau]\!]$ is associated to each op.

For each basic type τ, the reduction $\Downarrow_{[\![\tau]\!]}$ is a map in $\tau \times \mathcal{L}(\mathbb{N}) \to [\![\tau]\!]$. Intuitively, it maps an expression of type τ to its value in $[\![\tau]\!]$ for a given list l of pointers in memory. These reductions are defined in Figure 2, where e and d denote either an integer expression i or a boolean expression b.

$$\frac{(\text{e}, l) \Downarrow_{[\![\tau_1]\!]} m \qquad (\text{d}, l) \Downarrow_{[\![\tau_2]\!]} n}{(\text{e op d}, l) \Downarrow_{[\![\text{op}]\!]([\![\tau_1]\!],[\![\tau_2]\!])} [\![\text{op}]\!](m, n)}\ (\text{Op}) \qquad \frac{(\text{i}, l) \Downarrow_{\mathbb{Z}} n}{(\text{U}^f(\text{i}), l) \Downarrow_{\mathbb{C}^{2\times 2}} [\![\text{U}^f]\!](n)}\ (\text{Unit})$$

$$\frac{}{(n, l) \Downarrow_{\mathbb{Z}} n}\ (\text{Cst}) \qquad \frac{(\text{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} [x_1, \ldots, x_m] \qquad (\text{i}, l) \Downarrow_{\mathbb{Z}} k \in [1, m]}{(\text{s} \ominus [\text{i}], l) \Downarrow_{\mathcal{L}(\mathbb{N})} [x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_m]}\ (\text{Rm}_\in)$$

$$\frac{(\text{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} [x_1, \ldots, x_n]}{(|\text{s}|, l) \Downarrow_{\mathbb{Z}} n}\ (\text{Size}) \qquad \frac{(\text{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} [x_1, \ldots, x_m] \qquad (\text{i}, l) \Downarrow_{\mathbb{Z}} k \notin [1, m]}{(\text{s} \ominus [\text{i}], l) \Downarrow_{\mathcal{L}(\mathbb{N})} [\,]}\ (\text{Rm}_\notin)$$

$$\frac{}{(\text{nil}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} [\,]}\ (\text{Nil}) \qquad \frac{(\text{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} [x_1, \ldots, x_m] \qquad (\text{i}, l) \Downarrow_{\mathbb{Z}} k \in [1, m]}{(\text{s}[\text{i}], l) \Downarrow_{\mathbb{N}} x_k}\ (\text{Qu}_\in)$$

$$\frac{}{(\bar{\text{q}}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} l}\ (\text{Var}) \qquad \frac{(\text{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} [x_1, \ldots, x_m] \qquad (\text{i}, l) \Downarrow_{\mathbb{Z}} k \notin [1, m]}{(\text{s}[\text{i}], l) \Downarrow_{\mathbb{N}} 0}\ (\text{Qu}_\notin)$$

Fig. 2: Semantics of expressions

Note that in rule ($\text{Rm}_\notin$), if we try to delete an undefined index then we return the empty list, and in rule ($\text{Qu}_\notin$), if we try to access an undefined qubit index then we return the value 0 (defined indexes will always be positive). The standard gates $R_Y(\pi/4)$, $P(\pi/4)$, and $CNOT$, form a universal set of gates [4], which justifies the choice of NOT, $\text{R}_Y^f(\text{i})$, and $\text{Ph}^f(\text{i})$ as basic operators. For instance, we can simulate the application of an Hadamard gate H on q by the following statement q *= $\text{R}_Y^f(0)$; q *= NOT;, with the function f defined by

$\forall n, f(n) = \pi/4 \in [0, 2\pi) \cap \tilde{\mathbb{R}}$. By abuse of notation, we will sometimes use q *= H; to denote this statement. Using CNOT, we can also define the SWAP operation swapping the state between two qubits $\bar{q}[i]$ and $\bar{q}[j]$, with $i, j \in \mathbb{N},\ i \neq j$:

$$\mathrm{SWAP}(\bar{q}[i], \bar{q}[j]) \triangleq \mathrm{CNOT}(\bar{q}[i], \bar{q}[j])\ \mathrm{CNOT}(\bar{q}[j], \bar{q}[i])\ \mathrm{CNOT}(\bar{q}[i], \bar{q}[j]).$$

Let $\top$ and $\bot$ be two special symbols for termination and error, respectively, and let $\diamond$ stand for a symbol in $\{\top, \bot\}$. The set of *configurations* of dimension 2^n, denoted Conf_n, is defined by

$$\mathrm{Conf}_n \triangleq (\mathrm{Statements} \cup \{\top, \bot\}) \times \mathcal{H}_{2^n} \times \mathcal{P}(\mathbb{N}) \times \mathcal{L}(\mathbb{N}),$$

with $\mathcal{P}(\mathbb{N})$ being the powerset over $\mathbb{N}$. A configuration $c = (\mathrm{S}, |\psi\rangle, A, l) \in \mathrm{Conf}_n$ contains a statement S to be executed (provided that $\mathrm{S} \notin \{\top, \bot\}$), a quantum state $|\psi\rangle$ of length n, a set A containing the indexes of qubits that are allowed to be accessed by statement S, and a list l of qubit pointers.

The program big-step semantics $\longrightarrow$, described in Figure 3, is defined as a relation in $\bigcup_{n\in\mathbb{N}} \mathrm{Conf}_n \times \mathrm{Conf}_n$. In the rules of Figure 3, $\longrightarrow$ is annotated by an integer, called *level*. For example, the level of the conclusion in the ($\mathrm{Call}_{[\,]}$) rule is 1. The level is used to count the total number of procedure calls that are not in superposition (*i.e.*, in distinct branches of a quantum case).

We now give a brief intuition on the rules of Figure 3. Rules ($\mathrm{Asg}_\bot$) and ($\mathrm{Asg}_\top$) evaluate the application of a unitary operator, corresponding to $\mathrm{U}^f(\mathrm{j})$, to a qubit s[i]. For that purpose, they evaluate the index n of s[i] in the global memory. Rule ($\mathrm{Asg}_\bot$) deals with the error case, where the corresponding qubit is not allowed to be accessed. Rule ($\mathrm{Asg}_\top$) deals with the success case: the new quantum state is obtained by applying the result of tensoring the evaluation of $\mathrm{U}^f(\mathrm{j})$ to the right index. Rules ($\mathrm{Seq}_\diamond$) and ($\mathrm{Seq}_\bot$) evaluate the sequence of statements, depending on whether an error occurs or not. The (If) rule deals with classical conditionals in a standard way. The three rules ($\mathrm{Case}_\top$), ($\mathrm{Case}_\bot$), and ($\mathrm{Case}_\notin$) evaluate the qubit index n of the control qubit s[i]. Then they check whether this index belongs to the set of accessible qubits (is n in A?). If so, the two statements S_0 and S_1 are intuitively evaluated in superposition, on the projected state $\langle 0|_n|\psi\rangle$ and $\langle 1|_n|\psi\rangle$, respectively. During these evaluations, the index n cannot be accessed anymore. The rule ($\mathrm{Call}_{[\,]}$) treats the base case of a procedure call when the sorted set parameter is empty. In the non-empty case, rule ($\mathrm{Call}_\diamond$) evaluates the sorted set parameter s to l' and the integer parameter x to n. It returns the result of evaluating the procedure statement $\mathrm{S}^{\mathrm{proc}}\{n/\mathrm{x}\}$, where n has been substituted to x, w.r.t. the updated qubit pointers list l'.

For a given program P = D :: S and a given quantum state $|\psi\rangle \in \mathcal{H}_{2^n}$, the *initial configuration* for input $|\psi\rangle$ is $c_{init}(|\psi\rangle) \triangleq (\mathrm{S}, |\psi\rangle, \{1, \ldots, n\}, [1, \ldots, n]) \in \mathrm{Conf}_n$. A program is *error-free* if there is no initial configuration $c_{init}(|\psi\rangle)$ such that $c_{init}(|\psi\rangle) \longrightarrow (\bot, |\psi'\rangle, A, l)$. We write $[\![\mathrm{P}]\!](|\psi\rangle) = |\psi'\rangle$, whenever $c_{init}(|\psi\rangle) \xrightarrow{m} (\top, |\psi'\rangle, A, l)$ holds for some m. $(\top, |\psi'\rangle, A, l)$ is called a *terminal configuration*. Let $\mathcal{H} = \bigcup_n \mathcal{H}_{2^n}$, a program *terminates* if $[\![\mathrm{P}]\!]$ is a total function in $\mathcal{H} \to \mathcal{H}$. Note that if a program terminates then it is obviously error-free but the converse property does not hold. Every program P can be efficiently transformed into an error-free

$$\frac{}{(\mathbf{skip}, |\psi\rangle, A, l) \xrightarrow{0} (\top, |\psi\rangle, A, l)} \text{ (Skip)}$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \notin A}{(\mathrm{s[i]} \ \mathbf{*=}\ \mathrm{U}^f(\mathrm{j});, |\psi\rangle, A, l) \xrightarrow{0} (\bot, |\psi\rangle, A, l)} \text{ } (\mathrm{Asg}_\bot)$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \in A \qquad (\mathrm{U}^f(\mathrm{j}), l) \Downarrow_{\mathbb{C}^{2\times 2}} M}{(\mathrm{s[i]} \ \mathbf{*=}\ \mathrm{U}^f(\mathrm{j});, |\psi\rangle, A, l) \xrightarrow{0} (\top, I_{2^{n-1}} \otimes M \otimes I_{2^{l(|\psi\rangle)-n}} |\psi\rangle, A, l)} \text{ } (\mathrm{Asg}_\top)$$

$$\frac{(\mathrm{S}_1, |\psi\rangle, A, l) \xrightarrow{m_1} (\top, |\psi'\rangle, A, l) \qquad (\mathrm{S}_2, |\psi'\rangle, A, l) \xrightarrow{m_2} (\diamond, |\psi''\rangle, A, l)}{(\mathrm{S}_1 \ \mathrm{S}_2, |\psi\rangle, A, l) \xrightarrow{m_1+m_2} (\diamond, |\psi''\rangle, A, l)} \text{ } (\mathrm{Seq}_\diamond)$$

$$\frac{(\mathrm{S}_1, |\psi\rangle, A, l) \xrightarrow{m} (\bot, |\psi\rangle, A, l)}{(\mathrm{S}_1 \ \mathrm{S}_2, |\psi\rangle, A, l) \xrightarrow{m} (\bot, |\psi\rangle, A, l)} \text{ } (\mathrm{Seq}_\bot)$$

$$\frac{(\mathrm{b}, l) \Downarrow_{\mathbb{B}} b \in \mathbb{B} \qquad (\mathrm{S}_b, |\psi\rangle, A, l) \xrightarrow{m_b} (\diamond, |\psi'\rangle, A, l)}{(\mathbf{if}\ \mathrm{b}\ \mathbf{then}\ \mathrm{S}_{\mathbf{true}}\ \mathbf{else}\ \mathrm{S}_{\mathbf{false}}, |\psi\rangle, A, l) \xrightarrow{m_b} (\diamond, |\psi'\rangle, A, l)} \text{ (If)}$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \in A \qquad (\mathrm{S}_k, |\psi\rangle, A\backslash\{n\}, l) \xrightarrow{m_k} (\top, |\psi_k\rangle, A\backslash\{n\}, l)}{(\mathbf{qcase}\ \mathrm{s[i]}\ \mathbf{of}\ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}, |\psi\rangle, A, l) \xrightarrow{\max_k m_k} (\top, \sum_k |k\rangle_m\langle k|_n |\psi_k\rangle, A, l)} \text{ } (\mathrm{Case}_\top)$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \in A \qquad (\mathrm{S}_k, |\psi\rangle, A\backslash\{n\}, l) \xrightarrow{m_k} (\diamond_k, |\psi_k\rangle, A\backslash\{n\}, l) \qquad \bot \in \{\diamond_0, \diamond_1\}}{(\mathbf{qcase}\ \mathrm{s[i]}\ \mathbf{of}\ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}, |\psi\rangle, A, l) \xrightarrow{\max_k m_k} (\bot, |\psi\rangle, A, l)} \text{ } (\mathrm{Case}_\bot)$$

$$\frac{(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n \notin A}{(\mathbf{qcase}\ \mathrm{s[i]}\ \mathbf{of}\ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}, |\psi\rangle, A, l) \xrightarrow{0} (\bot, |\psi\rangle, A, l)} \text{ } (\mathrm{Case}_{\notin})$$

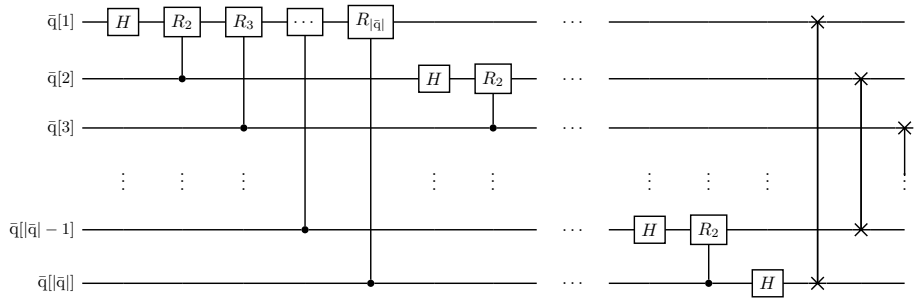
$$\frac{(\mathrm{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} l' \neq [\,] \qquad (\mathrm{i}, l) \Downarrow_{\mathbb{Z}} n \qquad (\mathrm{S}^{\mathrm{proc}}\{n/\mathrm{x}\}, |\psi\rangle, A, l') \xrightarrow{m} (\diamond, |\psi'\rangle, A, l')}{(\mathbf{call}\ \mathrm{proc[i](s)};, |\psi\rangle, A, l) \xrightarrow{m+1} (\diamond, |\psi'\rangle, A, l)} \text{ } (\mathrm{Call}_\diamond)$$

$$\frac{(\mathrm{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} [\,]}{(\mathbf{call}\ \mathrm{proc[i](s)};, |\psi\rangle, A, l) \xrightarrow{1} (\top, |\psi\rangle, A, l)} \text{ } (\mathrm{Call}_{[\,]})$$

Fig. 3: Semantics of statements

program $\mathrm{P}_{\neg\bot}$ such that $\forall|\psi\rangle$, if $[\![\mathrm{P}]\!](|\psi\rangle)$ is defined then $[\![\mathrm{P}]\!](|\psi\rangle) = [\![\mathrm{P}_{\neg\bot}]\!](|\psi\rangle)$. For example, an assignment s[i] $\mathbf{*=}$ $\mathrm{U}^f(\mathrm{j})$; can be transformed into the conditional statement **if** $((0 < \mathrm{i}) \wedge (\mathrm{i} \leq |\mathrm{s}|))$ **then** s[i] $\mathbf{*=}$ $\mathrm{U}^f(\mathrm{j})$; **else skip**;.

Example 1. A notable example of quantum algorithm is the Quantum Fourier Transform (QFT), used as a subroutine in Shor's algorithm [16], and whose quantum circuit is provided below, with $R_n \triangleq [\![\mathrm{Ph}^{\lambda x.\pi/2^{x-1}}]\!](n)$, for $n \geq 2$. After applying Hadamard and controlled R_n gates, the circuit performs a permutation of qubits using swap gates.



Note that $\lambda x.\pi/2^{x-1}$ is a total function in $\mathbb{Z} \to [0, 2\pi) \cap \tilde{\mathbb{R}}$. Hence, it is polynomial time approximable. The above circuit can be simulated for any number of qubits $|\mathrm{q}|$ by the following FOQ program QFT.

---

decl rec($\bar{\mathrm{p}}$){
  $\bar{\mathrm{p}}$[1] ***=** H;
  call rot[2]($\bar{\mathrm{p}}$);
  call rec($\bar{\mathrm{p}} \ominus [1]$); },

decl rot[x]($\bar{\mathrm{p}}$){
  if $|\bar{\mathrm{p}}| > 1$ **then**
    qcase $\bar{\mathrm{p}}$[2] **of** {
      $0 \to$ **skip**;
      $1 \to \bar{\mathrm{p}}[1]$ ***=** $\mathrm{Ph}^{\lambda x.\pi/2^{x-1}}$(x);
    }
    call rot[x + 1]($\bar{\mathrm{p}} \ominus [2]$);
  else skip; },

decl inv($\bar{\mathrm{p}}$){
  if $|\bar{\mathrm{p}}| > 1$ **then**
    SWAP($\bar{\mathrm{p}}[1], \bar{\mathrm{p}}[|\bar{\mathrm{p}}|]$);
    call inv($\bar{\mathrm{p}} \ominus [1, |\bar{\mathrm{p}}|]$);
  else skip; } ::

call rec($\bar{\mathrm{q}}$); **call** inv($\bar{\mathrm{q}}$);

Derivation tree and level. Given a configuration c wrt a fixed program P, $\pi_{\mathrm{P}} \rhd c$ denotes the *derivation tree* of P, the tree of root c whose children are obtained by applying the rules of Figures 2 and 3 on configuration c with respect to P. We write π instead of $\pi_{\mathrm{P}} \rhd c$ when P and c are clear from the context. Note that a derivation tree π can be infinite in the particular case of a non-terminating computation. When π' is finite, $\pi \trianglelefteq \pi'$ denotes that π is a subtree of π'.

In the case of a terminating computation $\pi \rhd c$, there exists a terminal configuration c' and a level $m \in \mathbb{N}$ such that $c \xrightarrow{m} c'$ holds. In this case, the level of π is defined as $\mathrm{lv}_\pi \triangleq m$. Given a FOQ program P that terminates, $\mathrm{level}_{\mathrm{P}}$ is a total function in $\mathbb{N} \to \mathbb{N}$ defined as $\mathrm{level}_{\mathrm{P}}(n) \triangleq \max_{|\psi\rangle \in \mathcal{H}_{2^n}} \mathrm{lv}_{\pi_{\mathrm{P}} \rhd c_{init}(|\psi\rangle)}$.

Intuitively, $\mathrm{level}_{\mathrm{P}}(n)$ corresponds to the maximal number of non-superposed procedure calls in any program execution on an input of length n.

Example 2. Consider the program QFT of example 1. Assume temporarily that QFT terminates (this will be shown in Example 3). For all $n \in \mathbb{N}$, $\text{level}_{\text{QFT}}(n) = \frac{(n+1)(n+2)}{2} + \lfloor \frac{n}{2} \rfloor + 1$. Indeed, on sorted sets of size n, procedure rec is called recursively $n+1$ times and makes $n+1$ calls to procedure rot on sorted sets of size n, $n-1$, ..., and 1. On sorted sets of size n, rot performs n recursive calls. Hence the total number of calls to rot is equal to $\sum_{i=1}^{n} i$. Finally, on a sorted set of size n, procedure inv does $\lfloor \frac{n}{2} \rfloor + 1$ recursive call.

A program P is reversible if it terminates and there exists a program P^{-1} such that $[\![\mathrm{P}^{-1}]\!] \circ [\![\mathrm{P}]\!] = Id$.

Theorem 1. *All terminating* FOQ *programs are reversible.*

3 Polynomial time soundness

In this section, we restrict the set of FOQ programs to a strict subset, named PFOQ, that is sound for the quantum complexity class FBQP. For this, we define two criteria: a criterion ensuring that a program terminates and a criterion preventing a terminating program from having an exponential runtime.

Polynomial-time FOQ. Given two statements S, S', we write $\mathrm{S} \in \mathrm{S}'$ to mean that S is a substatement of S' and $\mathrm{proc} \in \mathrm{S}$ holds if there are i and s such that **call** proc[i](s); $\in$ S. Given a program $\mathrm{P} = \mathrm{D} :: \mathrm{S}$, we define the relation $>_{\mathrm{P}} \subseteq$ Procedures $\times$ Procedures by $\mathrm{proc}_1 >_{\mathrm{P}} \mathrm{proc}_2$ if $\mathrm{proc}_2 \in \mathrm{S}^{\mathrm{proc}_1}$, for any two procedures proc_1, $\mathrm{proc}_2 \in$ S. Let the partial order $\geq_{\mathrm{P}}$ be the transitive and reflexive closure of $>_{\mathrm{P}}$ and define the equivalence relation $\sim_{\mathrm{P}}$ by $\mathrm{proc}_1 \sim_{\mathrm{P}} \mathrm{proc}_2$ if $\mathrm{proc}_1 \geq_{\mathrm{P}} \mathrm{proc}_2$ and $\mathrm{proc}_2 \geq_{\mathrm{P}} \mathrm{proc}_1$ both hold. Define also the strict order $>_{\mathrm{P}}$ by $\mathrm{proc}_1 >_{\mathrm{P}} \mathrm{proc}_2$ if $\mathrm{proc}_1 \geq_{\mathrm{P}} \mathrm{proc}_2$ and $\mathrm{proc}_1 \not\sim_{\mathrm{P}} \mathrm{proc}_2$ both hold.

Definition 1. *Let* WF *be the set of* FOQ *programs* P *that are error-free and satisfy the well-foundedness constraint:* $\forall \mathrm{proc} \in \mathrm{P},\ \forall$ **call** $\mathrm{proc}'[\mathrm{i}](\mathrm{s}); \in \mathrm{S}^{\mathrm{proc}}$,

$$\mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}' \Rightarrow \exists k > 0, \exists \mathrm{i}_1, \ldots, \mathrm{i}_k,\ s = \bar{\mathrm{p}} \ominus [\mathrm{i}_1, \ldots, \mathrm{i}_k].$$

Lemma 1 *If* $\mathrm{P} \in$ WF*, then* P *terminates.*

Example 3. Consider the program QFT of Example 1. The statements of the procedure declarations define the following relation: rec $>_{\text{QFT}}$ rec, rec $>_{\text{QFT}}$ rot, rot $>_{\text{QFT}}$ rot, and inv $>_{\text{QFT}}$ inv. Consequently, rec $\sim_{\text{QFT}}$ rec, rot $\sim_{\text{QFT}}$ rot, inv $\sim_{\text{QFT}}$ inv, and rec $>_{\text{QFT}}$ rot hold. For each call to an equivalent procedure, we check that the argument decreases: $\bar{\mathrm{p}} \ominus [1]$ in rec, $\bar{\mathrm{p}} \ominus [2]$ in rot, and $\bar{\mathrm{p}} \ominus [1, |\bar{\mathrm{p}}|]$ in inv. Consequently, QFT $\in$ WF. We deduce from Theorem 1 that QFT terminates.

We now add a further restriction on mutually recursive procedure calls for guaranteeing polynomial time using a notion of width.

Definition 2. *Given a program* P *and a procedure* $\mathrm{proc} \in \mathrm{P}$*, the* width *of* proc *in* P*, noted* $\mathrm{width}_{\mathrm{P}}(\mathrm{proc})$*, and the* width *of* proc *in* P relatively to statement S*, noted* $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S})$*, are two positive integers in* $\mathbb{N}$*. They are defined inductively by:*

$$
\begin{aligned}
&\mathrm{width}_{\mathrm{P}}(\mathrm{proc}) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}^{\mathrm{proc}}),\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{skip};) \triangleq 0,\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{q}\ \textbf{*=}\ \mathrm{U}^{f}(\mathrm{i});) \triangleq 0,\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1\ \mathrm{S}_2) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1) + w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_2),\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{if}\ \mathrm{b}\ \textbf{then}\ \mathrm{S}_{\textbf{true}}\ \textbf{else}\ \mathrm{S}_{\textbf{false}}) \triangleq \max(w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_{\textbf{true}}), w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_{\textbf{false}})),\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{qcase}\ \mathrm{q}\ \textbf{of}\ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}) \triangleq \max(w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0), w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1)),\\
&w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{call}\ \mathrm{proc}'[\mathrm{i}](\mathrm{s});) \triangleq \begin{cases} 1 & \textit{if } \mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}',\\ 0 & \textit{otherwise.} \end{cases}
\end{aligned}
$$

Definition 3 (PFOQ). *Let* PFOQ *be the set of programs* P *in* WF *that satisfy the following constraint:* $\forall \mathrm{proc} \in \mathrm{P}, \mathrm{width}_{\mathrm{P}}(\mathrm{proc}) \leq 1$.

Example 4. In the program of Example 1, $\mathrm{width}_{\mathrm{QFT}}(\mathrm{rec}) = \mathrm{width}_{\mathrm{QFT}}(\mathrm{rot}) = \mathrm{width}_{\mathrm{QFT}}(\mathrm{inv}) = 1$, since $\mathrm{rec} \succ_{\mathrm{QFT}} \mathrm{rot}$ holds. Since $\mathrm{QFT} \in \mathrm{WF}$, by Example 3, we conclude that QFT is a PFOQ program.

We now show that the level of a PFOQ program is bounded by a polynomial in the length of its input.

Lemma 2 *For each* PFOQ *program* P*, there exists a polynomial* $Q \in \mathbb{N}[X]$ *such that* $\forall n \in \mathbb{N},\ \mathrm{level}_{\mathrm{P}}(n) \leq Q(n)$.

Moreover, checking whether a program is PFOQ is tractable.

Theorem 2. *For each* FOQ *program* P*, it can be decided in time* $O(|\mathrm{P}|^2)$ *whether* $\mathrm{P}_{\neg\perp} \in \mathrm{PFOQ}$.

Quantum Turing machines and FBQP. Following Bernstein and Vazirani [3], a k-tape *Quantum Turing Machine* (QTM), with $k \geq 1$, is defined by a triplet (Σ, Q, δ) where Σ is a finite alphabet including a blank symbol $\#$, Q is a finite set of states with an initial state s_0 and a final state $s_\top \neq s_0$, and δ is the quantum transition function in $Q \times \Sigma^k \to \tilde{\mathbb{C}}^{Q \times \Sigma^k \times \{L,N,R\}^k}$; $\{L, N, R\}$ being the set of possible movements of a head on a tape. Each tape of the QTM is two-way infinite and contains cells indexed by $\mathbb{Z}$. A QTM successfully terminates if it reaches a superposition of only the final state $s_\top$. A QTM is said to be *well-formed* if the transition function δ preserves the norm of the superposition (or, equivalently, if the time evolution of the machine is unitary). The starting position of the tape heads is the *start cell*, the cell indexed by 0. If the machine terminates with all of its tape heads back on the start cells, it is called *stationary*. We will use *stationary* in the case where the machine terminates with its input tape head in the first cell, and all other tape heads in the last non-blank cell.

We will further refer to a QTM as being *in normal form* if the only transitions from the final state $s_\top$ are towards the initial state s_0. These will be important conditions for the composition and branching constructions of QTMs. If a QTM is well-formed, stationary, and in normal form, we will call it *conservative* [17] (N.B.: our notion of stationary QTM differs but can be shown to be equivalent to the definition of stationary QTM in [17]).

A configuration γ of a k-tape QTM is a tuple $(s, \overline{w}, \overline{n})$, where s is a state in Q, $\overline{w}$ is a k-tuple of words in Σ^*, and $\overline{n}$ is a k-tuple of indexes (head positions) in $\mathbb{Z}$. An initial (final) configuration γ_{init} (resp. γ_{fin}) is a configuration of the shape $(s_0, \overline{w}, \overline{0})$ (resp. $(s_\top, \overline{w}, \overline{0})$). We use $\gamma(w)$ to denote a configuration γ where the word w is written on the input/output tape. Following [3], we write $\mathcal{S}$ to represent the inner-product space of finite complex linear combinations of configurations of the QTM M with the Euclidean norm. A QTM M defines a linear time operator $U_M : \mathcal{S} \to \mathcal{S}$, that outputs a superposition of configurations $\sum_i \alpha_i |\gamma_i\rangle$ obtained by applying a single-step transition of M to a configuration $|\gamma\rangle$ (*i.e.*, $U_M|\gamma\rangle = \sum_i \alpha_i |\gamma_i\rangle$). Let U_M^t, for $t \geq 1$, be the t-steps transition obtained from U_M as follows: $U_M^1 \triangleq U_M$ and $U_M^{t+1} \triangleq U_M \circ U_M^t$. Given a quantum state $|\psi\rangle = \sum_{w \in \{0,1\}^n} \alpha_w |w\rangle$ and a configuration γ, let $\gamma(|\psi\rangle) \in \mathcal{S}$ be the quantum configuration defined by $\gamma(|\psi\rangle) \triangleq \sum_{w \in \{0,1\}^n} \alpha_w |\gamma(w)\rangle$.

A quantum function $f : \mathcal{H} \to \mathcal{H}$ is computed by the QTM M in time t if for any $|\psi\rangle \in \mathcal{H}$, $U_M^t(\gamma_{init}(|\psi\rangle)) = \gamma_{fin}(f(|\psi'\rangle))$. Given $T : \mathbb{N} \to \mathbb{N}$ and a quantum function f, we say that the QTM M computes f in time T if for inputs of length n, M computes f in time $T(n)$.

Definition 4. *Given two functions $f : \{0,1\}^\star \to \{0,1\}^\star$, $F : \mathcal{H} \to \mathcal{H}$, and a value $p \in [0,1]$, we say that f is computed by F with probability p if $\forall x \in \{0,1\}^\star$, $|\langle f(x)|F(|x\rangle)|^2 \geq p$.*

The class FBQP is the functional extension of the complexity class BQP.

Definition 5 ([3]). *A function $f \in \{0,1\}^\star \to \{0,1\}^\star$ is in FBQP iff there exist a QTM M and a polynomial $P \in \mathbb{N}[X]$ s.t. M computes f in time P with probability $\frac{2}{3}$.*

A function $f \in \{0,1\}^\star \to \{0,1\}^\star$ has a *polynomial bound* $P \in \mathbb{N}[X]$ if $\forall n \in \mathbb{N}, \forall x \in \{0,1\}^n, \exists k \leq P(n),\ f(x) \in \{0,1\}^k$. Functions in FBQP have a polynomial bound as the size of their output is smaller than the polynomial time bound.

Soundness. We show that QTMs can simulate the function computed by any terminating FOQ program. The time complexity of this simulation depends on the length of the input quantum state and on the level of the considered program.

Lemma 3 *For any terminating FOQ program P, there exists a conservative QTM M that computes $[\![\mathrm{P}]\!]$ in time $O(n + n \times \mathrm{level}_{\mathrm{P}}(n))$.*

Now we show that any PFOQ program computes a FBQP function.

Theorem 3. *Given a PFOQ program P, a function $f : \{0,1\}^\star \to \{0,1\}^\star$, and a value $p \in (\frac{1}{2}, 1]$. If f is computed by $[\![\mathrm{P}]\!]$ with probability p then $f \in$ FBQP.*

Proof. Using Lemma 2 and Lemma 3. □

4 FBQP completeness

In this section we show that any function in FBQP can be faithfully approximated by a PFOQ program. Toward this end, we show that Yamakami's [17] FBQP-complete function algebra can be exactly simulated in PFOQ.

Yamakami's function algebra. A characterization of FBQP was provided in [17] using a function algebra, named $\widehat{\square_1^{\mathrm{QP}}}$. Given a quantum state $|\psi\rangle$ and a word $w \in \{0,1\}^n$, with $n \leq l(|\psi\rangle)$. $|\psi\rangle$ can be written as $|\psi\rangle = \sum_i \alpha_i |w_i z_i\rangle$, with $w_i \in \{0,1\}^n$ and $z_i \in \{0,1\}^{l(|\psi\rangle)-n}$. We write $\langle w|\psi\rangle$ as an abuse of notation for the quantum state defined by $\langle w|\psi\rangle \triangleq \sum_i \alpha_i \langle w|w_i\rangle |z_i\rangle$.

Definition 6. $\widehat{\square_1^{\mathrm{QP}}}$ *is the smallest class of functions including the basic initial functions* $\{I, Ph_\theta, Rot_\theta, NOT, SWAP\}$*, with* $\theta \in [0, 2\pi) \cap \tilde{\mathbb{C}}$*,*

- $I(|\psi\rangle) \triangleq |\psi\rangle$
- $Ph_\theta(|\psi\rangle) \triangleq |0\rangle\langle 0|\psi\rangle + e^{i\theta}|1\rangle\langle 1|\psi\rangle$
- $Rot_\theta(|\psi\rangle) \triangleq \cos\theta|\psi\rangle + \sin\theta(|1\rangle\langle 0|\psi\rangle - |0\rangle\langle 1|\psi\rangle)$
- $NOT(|\psi\rangle) \triangleq |0\rangle\langle 1|\psi\rangle + |1\rangle\langle 0|\psi\rangle$
- $SWAP(|\psi\rangle) \triangleq \begin{cases} |\psi\rangle & \text{if } l(|\psi\rangle) \leq 1 \\ \sum_{a,b\in\{0,1\}} |ba\rangle\langle ab|\psi\rangle & \text{otherwise} \end{cases}$

and closed under schemes Comp, Branch, and $kQRec_t$*, for* $k, t \in \mathbb{N}$*,*

- $Comp[F,G](|\psi\rangle) \triangleq F(G(|\psi\rangle))$
- $Branch[F,G](|\psi\rangle) \triangleq \begin{cases} |\psi\rangle & \text{if } l(|\psi\rangle) \leq 1 \\ |0\rangle \otimes F(\langle 0|\psi\rangle) + |1\rangle \otimes G(\langle 1|\psi\rangle) & \text{otherwise} \end{cases}$
- $kQRec_t[F,G,H](|\psi\rangle) \triangleq \begin{cases} F(|\psi\rangle) & \text{if } l(|\psi\rangle) \leq t \\ G\left(\sum_{w\in\{0,1\}^k} |w\rangle \otimes F_w(\langle w|H(|\psi\rangle))\right) & \text{otherwise} \end{cases}$

  where each $F_w \in \{kQRec_t[F,G,H], I\}$*.*

To handle general FBQP functions, [17] defines the extended encoding of an input $x \in \{0,1\}^\star$ as $\phi_P(|x\rangle) \triangleq |0^{l(|x\rangle)}1\rangle|0^{P(l(|x\rangle))}10^{11P(l(|x\rangle))+6}1\rangle|x\rangle$, for some polynomial $P \in \mathbb{N}[X]$ that is an upper bound on the output size of the desired FBQP function. ϕ_P simply consists in the quantum state $|x\rangle$ preceded by a polynomial number of ancilla qubits. These ancilla provide space for internal computations and account for the polynomial bound associated to polynomial time QTMs.

Theorem 4 ([17]). *Given* $f : \{0,1\}^\star \to \{0,1\}^\star$ *with polynomial bound* $P \in \mathbb{N}[X]$*, the following statements are equivalent.*

1. *The function* f *is in* FBQP*.*
2. *There exists* $F \in \widehat{\square_1^{\mathrm{QP}}}$ *such that* $F \circ \phi_P$ *computes* f *with probability* $\frac{2}{3}$*.*

We show the following result by structural induction on a function in $\widehat{\square_1^{\mathrm{QP}}}$.

Theorem 5. *Let F be a function in $\widehat{\square_1^{\mathrm{QP}}}$. Then there exists a* PFOQ *program* P *such that* $[\![\mathrm{P}]\!] = F$.

We are now ready to state the completeness result.

Theorem 6. *For every function f in* FBQP *with polynomial bound $Q \in \mathbb{N}[X]$, there is a* PFOQ *program* P *such that* $[\![\mathrm{P}]\!] \circ \phi_Q$ *computes f with probability $\frac{2}{3}$.*

Proof. By Theorem 4 and Theorem 5. □

5 Compilation to polynomial-size quantum circuits

In this section, we provide an algorithm that compiles a PFOQ program on a given input length $n \in \mathbb{N}$ into a quantum circuit of size polynomial in n.

Quantum circuits [8] are a well-known graphical computational model for describing quantum computations. Qubits are represented by wires. Each unitary transformation U acting on n qubits can be represented as a gate U with n inputs and n outputs. A circuit C is an element of a PROP category ([11], a symmetric strict monoidal category) whose morphisms are generated by gates G and wires. Let $\mathbb{1}$ be the identity circuit (for any length) and $\circ$ and $\otimes$ be the composition and product, respectively. By abuse of notation, given k circuits $C^1, \ldots, C^k$, $\circ_{i=1}^k C^i$ will denote the circuit $\tilde{C^1} \circ \cdots \circ \tilde{C^k}$, where each circuit $\tilde{C^i}$ is obtained by tensoring C^i appropriately with identities so that the output of C^i matches the input of C^{i+1}. By construction, a circuit is acyclic. Each circuit C_n can be indexed by its number $n \in \mathbb{N}$ of input wires (i.e., non ancilla qubits) and computes a function $[\![C_n]\!] \in \mathcal{H}_{2^n} \to \mathcal{H}_{2^n}$. To deal with functions in $\mathcal{H} \to \mathcal{H}$, we consider families of circuits $(C_n)_{n\in\mathbb{N}}$, that are sequences of circuits such that each C_n encodes computation on quantum states of length n. Hence each circuit has n input qubits plus some extra ancilla qubits. These ancillas can be used to perform intermediate computations but also to represent functions whose output size is strictly greater than their input size. To avoid the consideration of families encoding undecidable properties, we put a uniformity restriction.

Definition 7. *A family of circuits $(C_n)_{n\in\mathbb{N}}$ is said to be* uniform *if there exists a polynomial time Turing machine that takes n as input and outputs a representation of C_n, for all $n \in \mathbb{N}$.*

In quantifying the complexity of a circuit, it is necessary to specify the considered elementary gates, and define the complexity of an operation as the number of elementary gates needed to perform it. In our setting, we consider the following set of universal elementary gates $\{R_Y(\pi/4), P(\pi/4), CNOT\}$. The size $\#C$ of a circuit C is equal to the number of its gates and wires.

Definition 8. *A family of circuits $(C_n)_{n\in\mathbb{N}}$ is said to be* polynomially-sized *with $\alpha \in \mathbb{N} \to \mathbb{N}$ ancilla qubits if there exists a polynomial $P \in \mathbb{N}[X]$ such that, for each $n \in \mathbb{N}$, $\#C_n \leq P(n)$ and the number of ancilla qubits in C_n is exactly $\alpha(n)$.*

Let $\chi_m : \mathcal{H}_{2^n} \to \mathcal{H}_{2^{n+m}}$ be defined by $\chi_m(|\psi\rangle) \triangleq |\psi\rangle \otimes |0^m\rangle$, for a state $|\psi\rangle$ of size n. Let $\xi_m : \mathcal{H}_{2^n} \to \mathcal{H}_{2^m}$, with $m \leq n$, be defined by $\xi_m(|\psi\rangle) \triangleq \sum_{w\in\{0,1\}^m} \sum_{z\in\{0,1\}^{n-m}} \langle wz|\psi\rangle\, |w\rangle$. Finally, let $|w|$, for $w \in \{0,1\}^\star$, be the size of the word w.

Theorem 1 (Adapted from [18] and [12]). *A function $f : \{0,1\}^\star \to \{0,1\}^\star$ is in* FBQP *iff there exists a uniform polynomially-sized family of circuits $(C_n)_{n\in\mathbb{N}}$ with α ancilla qubits s.t. $\forall x \in \{0,1\}^\star$, $\left|\left\langle f(x)\middle|\xi_{|f(x)|} \circ [\![C_{|x|}]\!] \circ \chi_{\alpha(|x|)}(|x\rangle)\right\rangle\right|^2 \geq \frac{2}{3}$.*

In Theorem 1, $[\![C_{|x|}]\!]$ is a function in $\mathcal{H}_{2^{|x|+\alpha(|x|)}} \to \mathcal{H}_{2^{|x|+\alpha(|x|)}}$ The function $\chi_{\alpha(|x|)}$ pads the input with ancilla in state $|0\rangle$ to match the circuit dimension. The function $\xi_{|f(x)|}$ projects the output of the circuit to match the length of the function output $|f(x)|$. Hence, for $|x\rangle \in \mathcal{H}_{2^{|x|}}$, $\xi_{|f(x)|} \circ [\![C_{|x|}]\!] \circ \chi_{\alpha(|x|)}(|x\rangle) \in \mathcal{H}_{2^{|f(x)|}}$.

Compilation to circuits. For each PFOQ program P, the existence of a polynomially-sized uniform family of circuits $(C_n)_{n\in\mathbb{N}}$ that computes $[\![\mathrm{P}]\!]$ is entailed by the combination of Lemma 2 and Theorem 1. However, due to the complex machinery of QTM, the constructions of both proofs cannot be used in practice to generate a circuit. In this section, we exhibit an algorithm that compiles directly a PFOQ program to a polynomially-sized circuit. Note that this compilation process requires some care since recursive procedure calls in quantum cases may yield an exponential number of calls. The remainder of this section will be devoted to present an algorithm, named **compile**, which, for a given PFOQ program P and a given integer n produces a circuit C_n such that $\forall |\psi\rangle \in \mathcal{H}_{2^n}$, $[\![\mathrm{P}]\!](|\psi\rangle) = \xi_n \circ [\![C_n]\!] \circ \chi_{\alpha(n)}(|\psi\rangle)$.

The **compile** algorithm uses two subroutines, named **compr** and **optimize**, and is defined by $\mathbf{compile}(\mathrm{P}, n) \triangleq \mathbf{compr}(\mathrm{P}, [1, \ldots, n], \cdot)$.

The subroutine **compr** (Algorithm 1) generates the circuit inductively on the program statement. It takes as inputs: a program P, a list of qubit pointers l, and a control structure cs. A *control structure* cs is a partial function in $\mathbb{N} \to \{0,1\}$, mapping a qubit pointer to a control value (of a quantum case). Let $\cdot$ be the control structure of empty domain. For $n \in \mathbb{N}$ and $k \in \{0,1\}$, $cs[n := k]$ is the control structure obtained from cs by setting $cs(n) \triangleq k$. For a given $x \in \{0,1\}^\star$, we say that state $|x\rangle$ *satisfies* cs if, $\forall n \in dom(cs)$, $cs(n) = k \Rightarrow |\langle k|_n |x\rangle|^2 = 1$. Two control structures cs and cs' are *orthogonal* if there does not exist a state $|x\rangle$ that satisfies cs and cs'. Note that if $\exists i \in dom(cs) \cap dom(cs')$, $cs(i) + cs'(i) = 1$ then cs and cs' are orthogonal.

Given a control structure cs and a statement S, a *controlled statement* is a pair $(cs, \mathrm{S}) \in \mathrm{Cst} \triangleq (\mathbb{N} \to \{0,1\}) \times \mathrm{Statements}$. Intuitively, a controlled statement (cs, S) denotes a statement controlled by the qubits whose indices are in $dom(cs)$. For a unitary gate $U \in \mathcal{H}_{2^n} \to \mathcal{H}_{2^n}$, a control structure cs, and a list of pointers $l = [x_1, \ldots, x_n] \in \mathcal{L}(\mathbb{N})$ such that $\{x_1, \ldots, x_n\} \cap dom(cs) = \varnothing$, $U(cs, l)$ denotes the circuit applying gate U on qubits $\bar{\mathrm{q}}[x_1], \ldots, \bar{\mathrm{q}}[x_n]$, whenever $\forall m \in dom(cs)$, $\bar{\mathrm{q}}[m]$ is in state $|cs(m)\rangle$. As demonstrated in [12], this circuit can be built with $O(card(dom(cs)))$ elementary gates and ancillas, and a single controlled-U gate.

Algorithm 1 (**compr**)
Input: $(\mathrm{P}, l, cs) \in \mathrm{Programs} \times \mathcal{L}(\mathbb{N}) \times (\mathbb{N} \to \{0,1\})$

Let $\mathrm{D} :: \mathrm{S} = \mathrm{P}$ **in**
if $\mathrm{S} =$ **skip**; **then**
  $C \leftarrow \mathbb{1}$ ▷ Identity circuit

else if $\mathrm{S} = \mathrm{s[i]}$ ***=** $\mathrm{U}^f(\mathrm{j})$; **and** $(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n$ **and** $(\mathrm{U}^f(\mathrm{j}), l) \Downarrow_{\mathbb{C}^{2\times 2}} M$ **then**
  $C \leftarrow M(cs, [n])$ ▷ Controlled gate

else if $\mathrm{S} = \mathrm{S}_1\,\mathrm{S}_2$ **then**
  $C \leftarrow$ **compr**$(\mathrm{D} :: \mathrm{S}_1, l, cs) \circ$ **compr**$(\mathrm{D} :: \mathrm{S}_2, l, cs)$ ▷ Composition

else if $\mathrm{S} =$ **if** b **then** $\mathrm{S}_{\mathbf{true}}$ **else** $\mathrm{S}_{\mathbf{false}}$ **and** $(\mathrm{b}, l) \Downarrow_{\mathbb{B}} b$ **then**
  $C \leftarrow$ **compr**$(\mathrm{D} :: \mathrm{S}_b, l, cs)$ ▷ Conditional

else if $\mathrm{S} =$ **qcase** s[i] **of** $\{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}$ **and** $(\mathrm{s[i]}, l) \Downarrow_{\mathbb{N}} n$ **then**
  $C \leftarrow$ **compr**$(\mathrm{D} :: \mathrm{S}_0, l, cs[n := 0]) \circ$ **compr**$(\mathrm{D} :: \mathrm{S}_1, l, cs[n := 1])$ ▷ Quantum case

else if $\mathrm{S} =$ **call** proc[i](s) **and** $(\mathrm{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} [\,]$ **then**
  $C \leftarrow \mathbb{1}$ ▷ Nil call

else if $\mathrm{S} =$ **call** proc[i](s) **and** $(\mathrm{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} l' \neq [\,]$ **and** $(\mathrm{i}, l) \Downarrow_{\mathbb{Z}} n$ **then**
  if $\mathrm{width}_{\mathrm{P}}(\mathrm{proc}) = 0$ **then**
    $C \leftarrow$ **compr**$(\mathrm{D} :: \mathrm{S}^{\mathrm{proc}}\{n/\mathrm{x}\}, l', cs)$ ▷ Non-recursive call
  else if $\mathrm{width}_{\mathrm{P}}(\mathrm{proc}) = 1$ **then**
    $C \leftarrow$ **optimize**$(\mathrm{D}, [(cs, \mathrm{S}^{\mathrm{proc}}\{n/\mathrm{x}\})], \mathrm{proc}, l', \{\})$ ▷ Recursive call
  end if
end if
return C

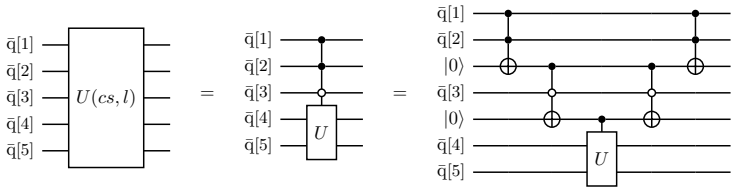


Fig. 4: Example of circuit $U(cs, l)$

Example 5. As an illustrative example, consider a binary gate U and a control structure cs such that $dom(cs) = \{1, 2, 3\}$, $cs(1) = cs(2) = 1$, and $cs(3) = 0$. Also consider a list $l = [4, 5] \in \mathcal{L}(\mathbb{N})$. The circuit $U(cs, l)$ is provided in Figure 4.

Similarly, we can define a generalized Toffoli gate as a circuit of the shape $NOT(cs, n)$. Since $card(dom(cs))$ will not scale with the size of the input, such a circuit has a constant cost in gates and ancillas and can thus be considered as an elementary gate. We will also be interested in rearranging wires under a given control structure. For two lists of qubit pointers $l_1 = [x_1, \ldots, x_n]$, $l_2 = [x'_1, \ldots, x'_n] \in \mathcal{L}(\mathbb{N})$, define $SWAP(cs, l_1, l_2)$ as the circuit that swaps the wires in l_1 with wires in l_2, controlled on cs. This circuit needs in the worst case one ancilla and $O(n)$ controlled $SWAP$ gates (also known as Fredkin gates).

Let $\mathcal{D} \triangleq \mathcal{D}(\text{Procedures} \times \mathbb{Z} \times \mathbb{N} \to \mathbb{N} \times \mathcal{L}(\mathbb{N}))$ denote the set of dictionaries mapping keys of the shape (proc, i, j) to pairs of the shape (a, l), where i is the value of a classical parameter, j is the size of a sorted set, and a is a qubit index. We will denote the empty dictionary by $\{\}$. Let also $a \leftarrow \textbf{new}\ ancilla()$ be an instruction that sets a to a fresh qubit index.

The subroutine **optimize** (Algorithm 2) treats the complex cases where circuit optimizations (merging) are needed, that is for recursive procedure calls. It takes as input a sequence of procedure declarations D, a list of controlled statements l_{Cst}, a procedure name proc, a list of qubit pointers l, and a dictionary Anc. The subroutine iterates on list l_{Cst} of controlled statements, indicating the statements left to be treated together with their control qubits. When recursive procedure calls appear in distinct branches of a quantum case, the algorithm merges these calls together. For that purpose, it uses new ancilla qubits as control qubits. Given procedure calls of shape **call** proc[i](s);, with respect to a given list $l \in \mathcal{L}(\mathbb{N})$, such that $(\text{i}, l) \Downarrow_{\mathbb{Z}} i$, $(\text{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} l'$, and $(|\text{s}|, l) \Downarrow_{\mathbb{N}} j$. If the key (proc, i, j) already exists in the dictionary Anc, the associated ancilla is re-used, otherwise, $\text{Anc}[\text{proc}, i, j]$ is set to (a, l').

Some extra ancillas e are also created for swapping wires and are not explicitly indexed since they are not revisited by the subroutine, and are just considered unique. Ancillas a and e are indexed and treated as input qubits, therefore they can be part of the domain of control structures.

Theorem 2. $\forall \text{P} \in \text{PFOQ}$, $\exists Q \in \mathbb{N}[X]$, $\forall n \in \mathbb{N}$, $\forall |\psi\rangle \in \mathcal{H}_{2^n}$, $[\![\text{P}]\!](|\psi\rangle) = \xi_n \circ [\![\textbf{compile}(\text{P}, n)]\!] \circ \chi_{\alpha(n)}(|\psi\rangle)$ *and* $\#\textbf{compile}(\text{P}, n) \leq Q(n)$.

Algorithm 2 (optimize) Build circuit for recursive procedure proc
Inputs: $(\mathrm{D}, l_{\mathrm{Cst}}, \mathrm{proc}, l, \mathrm{Anc}) \in \mathrm{Decl} \times \mathcal{L}(\mathrm{Cst}) \times \mathrm{Procedures} \times \mathcal{L}(\mathbb{N}) \times \mathcal{D}$

$C_{\mathrm{L}} \leftarrow \mathbb{1}$; $C_{\mathrm{R}} \leftarrow \mathbb{1}$; $\mathrm{P} \leftarrow \mathrm{D} :: \mathbf{skip}$;
while $l_{\mathrm{Cst}} \neq [\,]$ **do**
  $(cs, \mathrm{S}) \leftarrow hd(l_{\mathrm{Cst}})$; $l_{\mathrm{Cst}} \leftarrow tl(l_{\mathrm{Cst}})$

  if $\mathrm{S} = \mathrm{S}_1\,\mathrm{S}_2$ **then**
    if $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1) = 1$ **then**
      $l_{\mathrm{Cst}} \leftarrow l_{\mathrm{Cst}} @ [(cs, \mathrm{S}_1)]$; $C_{\mathrm{R}} \leftarrow \mathbf{compr}(\mathrm{D} :: \mathrm{S}_2, l, cs) \circ C_{\mathrm{R}}$
    else
      $l_{\mathrm{Cst}} \leftarrow l_{\mathrm{Cst}} @ [(cs, \mathrm{S}_2)]$; $C_{\mathrm{L}} \leftarrow C_{\mathrm{L}} \circ \mathbf{compr}(\mathrm{D} :: \mathrm{S}_1, l, cs)$
    end if
  end if

  if $\mathrm{S} = \mathbf{if}\ \mathrm{b}\ \mathbf{then}\ \mathrm{S}_{\mathbf{true}}\ \mathbf{else}\ \mathrm{S}_{\mathbf{false}}$ **and** $(\mathrm{b}, l) \Downarrow_{\mathbb{B}} b$ **then**
    if $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_b) = 1$ **then**
      $l_{\mathrm{Cst}} \leftarrow l_{\mathrm{Cst}} @ [(cs, \mathrm{S}_b)]$
    else
      $C_L \leftarrow C_L \circ \mathbf{compr}(\mathrm{D} :: \mathrm{S}_b, l, cs)$
    end if
  end if

  if $\mathrm{S} = \mathbf{qcase}\ s[\mathrm{i}]\ \mathbf{of}\ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}$ **and** $(s[\mathrm{i}], l) \Downarrow_{\mathbb{N}} n$ **then**
    if $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0) = 1$ **and** $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1) = 1$ **then**
      $l_{\mathrm{Cst}} \leftarrow l_{\mathrm{Cst}} @ [(cs[n := 0], \mathrm{S}_0), (cs[n := 1], \mathrm{S}_1)]$
    else if $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1) = 0$ **then**
      $l_{\mathrm{Cst}} \leftarrow l_{\mathrm{Cst}} @ [(cs[n := 0], \mathrm{S}_0)]$;
      $C_{\mathrm{R}} \leftarrow \mathbf{compr}(\mathrm{D} :: \mathrm{S}_1, l, cs[n := 1]) \circ C_{\mathrm{R}}$
    else if $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0) = 0$ **then**
      $l_{\mathrm{Cst}} \leftarrow l_{\mathrm{Cst}} @ [(cs[n := 1], \mathrm{S}_1)]$;
      $C_{\mathrm{R}} \leftarrow \mathbf{compr}(\mathrm{D} :: \mathrm{S}_0, l, cs[n := 0]) \circ C_{\mathrm{R}}$
    end if
  end if

  if $\mathrm{S} = \mathbf{call}\ \mathrm{proc}'[\mathrm{i}](\mathrm{s})$ **and** $(\mathrm{s}, l) \Downarrow_{\mathcal{L}(\mathbb{N})} l' \neq [\,]$ **and** $(\mathrm{i}, l) \Downarrow_{\mathbb{Z}} n$ **then**
    if $(\mathrm{proc}', n, |l'|) \in \mathrm{Anc}$ **then**
      Let $(a, l'') = \mathrm{Anc}[\mathrm{proc}', n, |l'|]$ **in**
      $e \leftarrow \mathbf{new}\ ancilla()$;
      $C_{\mathrm{L}} \leftarrow C_{\mathrm{L}} \circ NOT(cs, e) \circ NOT(\cdot[e = 1], a) \circ SWAP(\cdot[e = 1], l', l'')$;
      $C_{\mathrm{R}} \leftarrow SWAP(\cdot[e = 1], l'', l') \circ NOT(\cdot[e = 1], a) \circ NOT(cs, e) \circ C_{\mathrm{R}}$
    else
      $a \leftarrow \mathbf{new}\ ancilla()$
      $\mathrm{Anc}[\mathrm{proc}', n, |l'|] \leftarrow (a, l')$;
      $C_{\mathrm{L}} \leftarrow C_{\mathrm{L}} \circ NOT(cs, a)$; $C_{\mathrm{R}} \leftarrow NOT(cs, a) \circ C_{\mathrm{R}}$;
      $l_{\mathrm{Cst}} \leftarrow l_{\mathrm{Cst}} @ [(\cdot[a = 1], \mathrm{S}^{\mathrm{proc}'}\{n/\mathrm{x}\})]$
    end if
  end if
end while
return $C_L \circ C_R$

Example 6. **compile**(QFT, n) outputs the circuit provided in Example 1. Notice that there is no extra ancilla as no procedure call appears in the branch of a quantum case.

Polynomial-size circuits. We show Theorem 2 by exhibiting that any exponential growth of the circuit can be avoided by the **compile** algorithm using an argument based on orthogonal control structures. With a linear number of gates and a constant number of extra ancillas, we can merge calls referring to the same procedure, on different branches of a quantum case, when they are applied to sorted sets of equal size. An example of the construction is given in Figure 5 where two instances of a gate U are merged into one using $SWAP$ gates and gates controlled by orthogonal control structures.

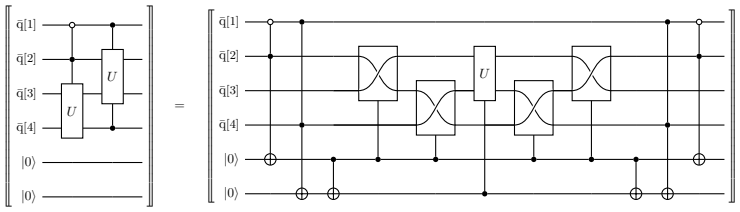


Fig. 5: Example of circuit optimization

The following proposition shows that multiple uses of a gate can be merged in one provided they are applied to orthogonal control structures.

Lemma 4 *For any circuit $C_n \triangleq \circ_{i=1}^{k} U(cs_i, l_i)$, with a unitary gate U, pairwise orthogonal $cs_1, \ldots, cs_k \in \mathrm{Cst}$, and $l_1, \ldots l_k \in \mathcal{L}(\mathbb{N})$, there exists a circuit C using one controlled gate U, $O(kn)$ gates, and $O(k)$ ancillas, and such that $[\![C]\!] = [\![C_n]\!]$.*

Now we show that orthogonality is an invariant property of **compile**.

Lemma 5 *Orthogonality is an invariant property of the control structures in l_{Cst} of the subroutine* **optimize**. *In other words, for any two distinct pairs (cs, S), (cs', S') in l_{Cst}, cs and cs' are orthogonal.*

Theorem 7. *For any program* P *in* PFOQ, **compile**(P, n) *runs in time* $O(n^{2|\mathrm{P}|+1})$.

Proof. Using Lemma 4 and Lemma 5. □

As there is no circuit duplication in the assignments of **compile**, we can deduce from Theorem 7 that the compiled circuit is of polynomial size.

Corollary 1. *For any program* P *in* PFOQ, *there exists a polynomial $Q \in \mathbb{N}[X]$ such that* #**compile**(P, n) $\leq Q(n)$.

References

1. Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions. *computational complexity*, 2(2):97–110, Jun 1992.
2. Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.
3. Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.
4. P. Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. On universal and fault-tolerant quantum computing, 1999.
5. Hans J Briegel, David E Browne, Wolfgang Dür, Robert Raussendorf, and Maarten Van den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, 2009.
6. Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. Quantum implicit computational complexity. *Theoretical Computer Science*, 411(2):377–409, 2010.
7. Vincent Danos and Elham Kashefi. Determinism in the one-way model. *Physical Review A*, 74(5):052310, 2006.
8. David Elieser Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425(1868):73–90, 1989.
9. Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, Jun 1982.
10. Ugo Dal Lago. A short introduction to implicit computational complexity. In *ESSLLI 2010*, pages 89–109, 2011.
11. Saunders MacLane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40–106, 1965.
12. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011.
13. Romain Péchoux. Implicit computational complexity: past and future. Mémoire d'habilitation à diriger des recherches, 2020. Université de Lorraine.
14. Neil J. Ross. Algebraic and logical methods in quantum computation. PhD thesis, 2015.
15. Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.
16. Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.
17. Tomoyuki Yamakami. A schematic definition of quantum polynomial time computability. *J. Symb. Log.*, 85(4):1546–1587, 2020.
18. Andrew Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 352–361, 1993.

A Proofs

Theorem 1. *All terminating* FOQ *programs are reversible.*

Proof. We show this by induction on the structure of the program. Let $\mathrm{P} = \mathrm{D} :: \mathrm{S}$ be a program that terminates. Define $\mathrm{P}^{-1} \triangleq (\mathrm{D} :: \mathrm{S})^{-1}$ inductively by

$$\begin{aligned}
(\mathrm{D} :: \mathrm{S})^{-1} &\triangleq \mathrm{D}^{-1} :: \mathrm{S}^{-1} \\
(\textbf{decl } \mathrm{proc[x]}(\bar{\mathrm{p}})\{\mathrm{S}\}, \mathrm{D})^{-1} &\triangleq \textbf{decl } \mathrm{proc[x]}(\bar{\mathrm{p}})\{\mathrm{S}^{-1}\}, \mathrm{D}^{-1} \\
\varepsilon^{-1} &\triangleq \varepsilon \\
\textbf{skip;}^{-1} &\triangleq \textbf{skip;} \\
(\mathrm{q} \mathrel{*}\!\!= \mathrm{U}^f(\mathrm{i});)^{-1} &\triangleq \mathrm{q} \mathrel{*}\!\!= (\mathrm{U}^f(\mathrm{i}))^\dagger; \\
(\mathrm{S}_1\ \mathrm{S}_2)^{-1} &\triangleq \mathrm{S}_2^{-1}\ \mathrm{S}_1^{-1} \\
(\textbf{if } \mathrm{b} \textbf{ then } \mathrm{S}_{\textbf{true}} \textbf{ else } \mathrm{S}_{\textbf{false}})^{-1} &\triangleq \textbf{if } \mathrm{b} \textbf{ then } \mathrm{S}^{-1}_{\textbf{true}} \textbf{ else } \mathrm{S}^{-1}_{\textbf{false}} \\
(\textbf{qcase } \mathrm{q} \textbf{ of } \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\})^{-1} &\triangleq \textbf{qcase } \mathrm{q} \textbf{ of } \{0 \to \mathrm{S}_0^{-1}, 1 \to \mathrm{S}_1^{-1}\} \\
(\textbf{call } \mathrm{proc[i]}(\mathrm{s});)^{-1} &\triangleq \textbf{call } \mathrm{proc[i]}(\mathrm{s});,
\end{aligned}$$

with $\mathrm{NOT}^\dagger \triangleq \mathrm{NOT}$, $\mathrm{R}^f_Y(\mathrm{i})^\dagger \triangleq \mathrm{R}^{-f}_Y(\mathrm{i})$ and $\mathrm{Ph}^f(\mathrm{i})^\dagger \triangleq \mathrm{Ph}^{-f}(\mathrm{i})$. P^{-1} terminates and is such that $\forall |\psi\rangle,\ [\![\mathrm{P}^{-1}]\!]([\![\mathrm{P}]\!](|\psi\rangle)) = |\psi\rangle$. □

Lemma 1 *If* $\mathrm{P} \in \mathrm{WF}$*, then* P *terminates.*

Proof. For a given program P, we define a lexicographical partial order between configurations whose statements are procedure calls.

$$(\textbf{call } \mathrm{proc[i]}(\mathrm{s});, |\psi\rangle, A, l) \gg_\mathrm{P} (\textbf{call } \mathrm{proc}'\mathrm{[i']}(\mathrm{s'});, |\psi'\rangle, A', l')$$

if $(\mathrm{proc} >_\mathrm{P} \mathrm{proc}') \vee (\mathrm{proc} \sim_\mathrm{P} \mathrm{proc}' \wedge n_\mathrm{s} > n_{\mathrm{s}'})$, provided that $(|\mathrm{s}|, l) \Downarrow_\mathbb{N} n_\mathrm{s}$ and $(|\mathrm{s}'|, l') \Downarrow_\mathbb{N} n_{\mathrm{s}'}$.

Given $\pi \rhd (\textbf{call } \mathrm{proc[i]}(\mathrm{s});, |\psi\rangle, A, l)$ and $\pi' \rhd (\textbf{call } \mathrm{proc}'\mathrm{[i']}(\mathrm{s'});, |\psi'\rangle, A', l')$, we show that if $\pi' \trianglelefteq \pi$ then it holds that

$$(\textbf{call } \mathrm{proc[i]}(\mathrm{s});, |\psi\rangle, A, l) \gg_\mathrm{P} (\textbf{call } \mathrm{proc}'\mathrm{[i']}(\mathrm{s'});, |\psi'\rangle, A', l').$$

Consider a program $\mathrm{P} \in \mathrm{WF}$. Assume that $\pi' \trianglelefteq \pi$, then it holds that $\mathrm{proc} \geq_\mathrm{P} \mathrm{proc}'$. Hence, either $\mathrm{proc} >_\mathrm{P} \mathrm{proc}'$ or $\mathrm{proc} \sim_\mathrm{P} \mathrm{proc}'$. In this latter case, by transitivity of $\sim_\mathrm{P}$, $\mathrm{s}' = \bar{\mathrm{p}} \ominus [\mathrm{i}_1, \ldots, \mathrm{i}_k]$, for some $k > 0$. It implies that the evaluation of $|\mathrm{s}'|$ is strictly smaller than the evaluation of $|\mathrm{s}|$, by transitivity. We conclude by observing that $\gg_\mathrm{P}$ is a well-founded order. □

Lemma 2 *For each* PFOQ *program* P*, there exists a polynomial* $Q \in \mathbb{N}[X]$ *such that* $\forall n \in \mathbb{N},\ \mathrm{level}_\mathrm{P}(n) \leq Q(n)$.

Proof. Consider a PFOQ program $\mathrm{P} = \mathrm{D} :: \mathrm{S}$. We define the *rank* of a procedure in P as follows:

$$\begin{aligned}
&rk(\mathrm{proc}) \triangleq 0 && \text{if } \nexists\, \mathrm{proc}',\ \mathrm{proc} >_\mathrm{P} \mathrm{proc}', \\
&rk(\mathrm{proc}) \triangleq \max\{rk(\mathrm{proc}') + 1 \mid \mathrm{proc} >_\mathrm{P} \mathrm{proc}'\} && \text{otherwise.}
\end{aligned}$$

The rank of program P is defined by $rk(\mathrm{P}) \triangleq \max_{\mathrm{proc}\in\mathrm{P}} rk(\mathrm{proc})$.

We show the result by induction on the rank of procedure calls in S. Take **call** proc[i](s); $\in$ S. If $rk(\mathrm{proc}) = 0$ and the procedure is not recursive then there is only 1 call to a procedure. If the procedure is recursive, it can be called at most once in each branch of a quantum case statement. Hence there can be at most $n+1$ such calls in the full quantum case branch of the derivation tree and it holds that $\mathrm{level}_{\mathrm{D}::\mathbf{call}\ \mathrm{proc[i](s);}}(n) = O(n)$.

Induction hypothesis: assume that any procedure proc′ such that $rk(\mathrm{proc}') \leq k$ satisfies $\mathrm{level}_{\mathrm{D}::\mathbf{call}\ \mathrm{proc'[i](s);}}(n) = O(n^{k+1})$. Consider a procedure proc such that $rk(\mathrm{proc}) = k+1$. If the procedure is not recursive then it can call a constant number (bounded by the size of the program) of procedures of strictly smaller rank. By induction hypothesis,

$$\mathrm{level}_{\mathrm{D}::\mathbf{call}\ \mathrm{proc[i](s);}}(n) = \sum_{\mathrm{proc}'<_{\mathrm{P}}\mathrm{proc}} O(n^{rk(\mathrm{proc}')+1}) = O(n^{rk(\mathrm{proc})}).$$

If the procedure is recursive, it can be called at most once in each branch of a quantum case statement. Hence there can be at most $n+1$ such calls in the full quantum case branch of the derivation tree. Moreover, each of these calls can perform a constant number of calls to procedures of strictly smaller rank. Consequently,

$$\mathrm{level}_{\mathrm{D}::\mathbf{call}\ \mathrm{proc[i](s);}}(n) = O(n) + \sum_{i=0}^{n} \sum_{\mathrm{proc}'<_{\mathrm{P}}\mathrm{proc}} O(n^{rk(\mathrm{proc}')+1}) = O(n^{rk(\mathrm{proc})+1}).$$

We conclude by observing that for a program $\mathrm{P} = \mathrm{D} :: \mathrm{S}_1 \ \ldots \ \mathrm{S}_k$, it holds that $level_{\mathrm{P}}(n) = O(\sum_{i=1}^{k} \mathrm{level}_{\mathrm{D}::\mathrm{S}_i}(n)) = O(n^{rk(\mathrm{P})+1})$. □

Theorem 2. *For each* FOQ *program* P*, it can be decided in time* $O(|\mathrm{P}|^2)$ *whether* $\mathrm{P}_{\neg\perp} \in$ PFOQ*.*

Proof. The relations $\sim_{\mathrm{P}}$, $\geq_{\mathrm{P}}$, and $>_{\mathrm{P}}$ can be computed in time $O(|\mathrm{P}|^2)$. Moreover, any FOQ program P can be transformed in time $O(|\mathrm{P}|)$ into an equivalent error-free program $\mathrm{P}_{\neg\perp}$ of size $O(|\mathrm{P}|)$. Consequently, it can be decided in time $O(|\mathrm{P}|^2)$ whether $\mathrm{P}_{\neg\perp} \in$ WF and each width can be computed in time $O(|\mathrm{P}|^2)$. □

Lemma 3 *For any terminating* FOQ *program* P*, there exists a conservative QTM* M *that computes* $[\![\mathrm{P}]\!]$ *in time* $O(n + n \times \mathrm{level}_{\mathrm{P}}(n))$*.*

Proof. Consider a terminating FOQ program $\mathrm{P} = \mathrm{D} :: \mathrm{S}$. We build a 4-tape QTM M computing $[\![\mathrm{P}]\!]$ inductively on the statement S. Fix $\Sigma \triangleq \{0, 1, \#, \|, \&\}$, where # is the blank symbol and where $\|$ and & are special separation symbols for encoding stacks. The input tape t_{in} of M contains a word in $\{0,1,\#\}^n$ encoding the quantum state. The 3 working tapes are t_{call}, t_l, and $t_{\mathbb{K}}$ for storing the integer values of a procedure call, the list of qubit pointers, and intermediate classical computations, respectively, as words in Σ^*. The configurations of M will be in $Q \times (\Sigma^*)^4 \times \mathbb{Z}^4$, for some finite set of states Q. In particular, the initial

configuration is $(s_0, w, \varepsilon, \varepsilon, \varepsilon, 0, 0, 0, 0)$, with $w \in \{0,1\}^n$ encoding a quantum state of length $n \in \mathbb{N}$; the tapes t_{call}, t_l, and $t_{\mathbb{K}}$ are initially empty (ε). The tape heads all start on the first cells indexed by 0. For $m \in \mathbb{Z}$, let $t(m)$ denote the symbol at position m on tape t. Given a word $w \in \Sigma^*$ and a tape t, tw denotes that the content of t ends with the word w.

By abuse of notation, let $[\![\mathrm{e}]\!]$ denote the result of evaluating the expression e with respect to the machine current configuration. Also, we will assume that deterministic computations, such as taking tape $t \| [\![\mathrm{i}]\!]$ and appending $f([\![\mathrm{i}]\!])$, for any function f, are done by a reversible Turing machine [2], as reversible TMs are well-formed QTMs [3, Theorem 4.2].

We now describe a QTM M simulating P inductively on the statement S. The **skip**; statement is trivial. If S = q *= U^f(j);, M appends $[\![\mathrm{q}]\!]$ to $t_{\mathbb{K}}$. As the program terminates, $t_{in}([\![\mathrm{q}]\!]) \neq \#$ and the transition function is set to:

$$\forall a \in \{0,1\},\ \delta(s_{\mathrm{S}}, t_{in}([\![\mathrm{q}]\!]), s_{next(\mathrm{S})}, a, N) \triangleq \langle t_{in}([\![\mathrm{q}]\!]) | [\![\mathrm{U}^f]\!]([\![\mathrm{j}]\!]) | a \rangle$$

where s_{S} is the state before executing the assignment when the head of the input tape has been moved to position $[\![\mathrm{q}]\!]$, and $s_{next(\mathrm{S})}$ is the state just after executing the assignment. Finally, the machine erases $\| [\![\mathrm{q}]\!]$ at the end of $t_{\mathbb{K}}$, leaves its head in the last non-blank cell of $t_{\mathbb{K}}$, and moves the head in t_{in} back to the initial cell. Program P has $\mathrm{level}_{\mathrm{P}}(n) = 0$ and the simulating machine runs in time $O(n)$.

For the remaining statements, assume by induction hypothesis the existence of two conservative QTMs M_1 and M_2 that compute functions $[\![\mathrm{P}_1]\!]$ and $[\![\mathrm{P}_2]\!]$, respectively, with $\mathrm{P}_1 \triangleq \mathrm{D} :: \mathrm{S}_1$ and $\mathrm{P}_2 \triangleq \mathrm{D} :: \mathrm{S}_2$. By induction hypothesis M_1 and M_2 run in time $O(n + n \times \mathrm{level}_{\mathrm{P}_i}(n))$. States of M_i will be denoted by s^i, for $i \in \{1,2\}$. By using internal clocks, we can assume without loss of generality that machines M_i halt in exactly the same time for any quantum input of length n.

Consider the case S = $\mathrm{S}_1\,\mathrm{S}_2$. Machine M is defined as in [3, Dovetailing Lemma], with the initial state $s_0 \triangleq s_0^1$, its final state $s_\top \triangleq s_\top^2$, and the two machines are composed by setting $s_\top^1 = s_0^2$. The machine M is stationary, well-formed and it is well-behaved since the running time of M_2 only depends on n and the output of M_1 contains a superposition of equally sized quantum states. M computes $[\![\mathrm{P}]\!]$ in time $O(n + n \times \mathrm{level}_{\mathrm{P}_1}(n)) + O(n + n \times \mathrm{level}_{\mathrm{P}_2}(n)) = O(n + n \times \mathrm{level}_{\mathrm{P}}(n))$.

For the conditional S = **if** b **then** S_1 **else** S_2, we build a machine M that concatenates $\| [\![\mathrm{b}]\!]$ on the working tape $t_{\mathbb{K}}$ and runs M_1 or M_2 deterministically depending on the value of $[\![\mathrm{b}]\!]$, using the [3, Branching Lemma]. Then we erase $[\![\mathrm{b}]\!]$ from the end of tape $t_{\mathbb{K}}$. M computes $[\![\mathrm{P}]\!]$ in time $\max_i(O(n + n \times \mathrm{level}_{\mathrm{P}_i}(n))) = O(n + n \times \mathrm{level}_{\mathrm{P}}(n))$.

For the quantum case S = **qcase** q **of** $\{0 \to \mathrm{S}_1, 1 \to \mathrm{S}_2\}$, the machine appends $\| [\![\mathrm{q}]\!]$ on tape $t_{\mathbb{K}}$. It reads $t_{in}([\![\mathrm{q}]\!])$, sets it to #, and if it reads 0 runs M_1, if it reads 1, runs M_2. Finally, from state $s_\top^1$, it writes 0 in $t_{in}([\![\mathrm{q}]\!])$, moves the head to index 0, and transitions to $s_\top$; similarly, from state $s_\top^2$, the machines writes 1 in $t_{in}([\![\mathrm{q}]\!])$ before moving the head and transitioning to $s_\top$. We have that M computes $[\![\mathrm{P}]\!]$ in time $\max_i(O(n + n \times \mathrm{level}_{\mathrm{P}_i}(n))) = O(n + n \times \mathrm{level}_{\mathrm{P}}(n))$.

For the procedure call S = **call** proc[i](s);, inductively define machine M as follows: update t_{call} by appending $[\![\mathrm{i}]\!]$, and update t_l by adding the qubit pointer

indices excluded in $[\![\mathrm{s}]\!]$, separating them using &. We then run machine M_{proc} that computes the function $[\![\mathrm{P}_{\mathrm{proc}}]\!]$, for $\mathrm{P}_{\mathrm{proc}} \triangleq \mathrm{D} :: \mathrm{S}^{\mathrm{proc}}\{[\![\mathrm{i}]\!]/\mathrm{x}, \mathrm{s}/\bar{\mathrm{q}}\}$, in time $O(m + m \times \mathrm{level}_{\mathrm{P}_{\mathrm{proc}}}(m))$, with $m \triangleq [\![|\mathrm{s}|]\!] \leq n$, afterwards erasing $[\![\mathrm{i}]\!]$ and the new indices of t_{in}. As $\mathrm{level}_{\mathrm{P}_{\mathrm{proc}}}(n) = O(\mathrm{level}_{\mathrm{P}}(n))$ and the complexity of M is $O(n + n \times \mathrm{level}_{\mathrm{P}}(n))$. This concludes the proof. □

Theorem 5. *Let F be a function in $\widehat{\square_1^{\mathrm{QP}}}$. Then there exists a* PFOQ *program* P *such that* $[\![\mathrm{P}]\!] = F$.

Proof. We prove this result by structural induction on a function in the $\widehat{\square_1^{\mathrm{QP}}}$ algebra. The basic initial function I can be simulated by $\mathrm{P}(\bar{\mathrm{q}}) = \varepsilon :: \mathbf{skip};$. $F \in \{Ph_\theta, ROT_\theta, NOT\}$ can be simulated using an assignment. In these cases $\mathrm{P}(\bar{\mathrm{q}}) = \varepsilon :: \bar{\mathrm{q}}[1]$ ***=** $\mathrm{U}^f(0)$; with f such that $[\![\mathrm{U}^f]\!](0) = F$. The basic initial function $SWAP$ can be simulated by the program $\mathrm{P}(\bar{\mathrm{q}}) = \varepsilon :: \mathrm{SWAP}(\bar{\mathrm{q}}[1], \bar{\mathrm{q}}[2])$, with the SWAP statement defined in Section 2.

We now simulate the $Comp$, $Branch$ and $kQRec_t$ schemes. For that purpose, assume the existence of PFOQ programs $\mathrm{P}_F(\bar{\mathrm{q}}) = \mathrm{D}_F :: \mathrm{S}_F$, $\mathrm{P}_G(\bar{\mathrm{q}}) = \mathrm{D}_G :: \mathrm{S}_G$, and $\mathrm{P}_H(\bar{\mathrm{q}}) = \mathrm{D}_H :: \mathrm{S}_H$ simulating the $\widehat{\square_1^{\mathrm{QP}}}$ functions F, G, and H, respectively. For simplicity, we assume that there are no name clashes between the procedures declared in D_F, D_G, and D_H.

$Comp[F, G]$ can be simulated by $\mathrm{P}(\bar{\mathrm{q}}) = \mathrm{D}_F, \mathrm{D}_G :: \mathrm{S}_G\ \mathrm{S}_F$. Moreover $\mathrm{P} \in$ PFOQ by construction.

$Branch[F, G]$ can be simulated by

$$\begin{aligned}\mathrm{P}(\bar{\mathrm{q}}) =& \mathrm{D}_F, \mathrm{D}_G, \mathbf{decl}\ \mathrm{proc}_F(\bar{\mathrm{q}})\{\mathrm{S}_F\},\ \mathbf{decl}\ \mathrm{proc}_G(\bar{\mathrm{q}})\{\mathrm{S}_G\} ::\\ &\mathbf{qcase}\ \bar{\mathrm{q}}[1]\ \mathbf{of}\ \{0 \to \mathbf{call}\ \mathrm{proc}_F(\bar{\mathrm{q}} \ominus [1]);, 1 \to \mathbf{call}\ \mathrm{proc}_G(\bar{\mathrm{q}} \ominus [1]);\}.\end{aligned}$$

Procedures proc_F and proc_G are not recursive and therefore $\mathrm{P} \in$ PFOQ.

We introduce the following syntactical sugar for an extended quantum case for $k \geq 2$, $\forall a \in \{0,1\}$, and $\forall w \in \{0,1\}^{k-1}$:

$$\begin{aligned}&\mathbf{qcase}\ \mathrm{s}[\mathrm{i}_1, \ldots, \mathrm{i}_k]\ \ \mathbf{of}\ \{aw \to \mathrm{S}_{aw}\}\\ \triangleq\ &\mathbf{qcase}\ \mathrm{s}[\mathrm{i}_1]\ \mathbf{of}\ \{\ 0 \to \mathbf{qcase}\ \mathrm{s}[\mathrm{i}_2 \ldots \mathrm{i}_k]\ \mathbf{of}\ \{w \to \mathrm{S}_{0w}\},\\ &\qquad\qquad 1 \to \mathbf{qcase}\ \mathrm{s}[\mathrm{i}_2 \ldots \mathrm{i}_k]\ \mathbf{of}\ \{w \to \mathrm{S}_{1w}\}\}.\end{aligned}$$

$kQRec_t[F,G,H]$ can be simulated by

$$
\begin{aligned}
&\mathrm{P}(\bar{\mathrm{q}}) = \mathrm{D}_F,\ \mathrm{D}_G,\ \mathrm{D}_H,\\
&\quad \textbf{decl}\ \mathrm{proc}_F(\bar{\mathrm{q}})\{\mathrm{S}_F\},\ \textbf{decl}\ \mathrm{proc}_G(\bar{\mathrm{q}})\{\mathrm{S}_G\},\ \textbf{decl}\ \mathrm{proc}_H(\bar{\mathrm{q}})\{\mathrm{S}_H\},\\
&\quad \textbf{decl}\ \mathrm{kQRec}_t(\bar{\mathrm{p}})\{\\
&\qquad \textbf{if}\ |\bar{\mathrm{p}}| > t\ \textbf{then}\\
&\qquad\quad \textbf{call}\ \mathrm{proc}_H(\bar{\mathrm{p}});\\
&\qquad\quad \textbf{qcase}\ \bar{\mathrm{p}}[1\ldots k]\ \textbf{of}\ \{w \to \mathrm{S}_w\}\\
&\qquad\quad \textbf{call}\ \mathrm{proc}_G(\bar{\mathrm{p}});\\
&\qquad \textbf{else}\\
&\qquad\quad \textbf{call}\ \mathrm{proc}_F(\bar{\mathrm{p}});\}::\\
&\quad \textbf{call}\ \mathrm{kQRec}_t(\bar{\mathrm{q}});
\end{aligned}
$$

$$
\text{with } \mathrm{S}_w \triangleq \begin{cases} \textbf{call}\ \mathrm{kQRec}_t(\bar{\mathrm{p}} \ominus [1\ldots k]); & \text{if } F_w = kQRec_t,\\ \textbf{skip}; & \text{if } F_w = I. \end{cases}
$$

The recursive calls to procedure kQRec_t are of the shape **call** $\mathrm{kQRec}_t(\bar{\mathrm{p}} \ominus [1,\ldots,k])$;. More $\mathrm{kQRec}_t >_{\mathrm{P}} \mathrm{proc}_F, \mathrm{proc}_G, \mathrm{proc}_H$, and $\mathrm{P}_F,\ \mathrm{P}_G,\ \mathrm{P}_H \in \mathrm{WF}$. Consequently, $\mathrm{P} \in \mathrm{WF}$. Finally, the procedure kQRec_t is called recursively at most once per branch of a quantum case. Consequently, $width_{\mathrm{P}}(\mathrm{kQRec}_t) \leq 1$ and P is a PFOQ program. □

Theorem 3. *Given a* PFOQ *program* P*, a function* $f : \{0,1\}^\star \to \{0,1\}^\star$*, and a value* $p \in (\frac{1}{2}, 1]$*. If* f *is computed by* $[\![\mathrm{P}]\!]$ *with probability* p *then* $f \in$ FBQP*.*

Proof. Given a PFOQ program P, by Lemma 3, there exists a conservative QTM M_{P} that computes $[\![\mathrm{P}]\!]$ in time $O(n + n \times \mathrm{level}_{\mathrm{P}}(n))$. By Lemma 2, there exists a polynomial $Q \in \mathbb{N}[X]$ such that $\forall n,\ \mathrm{level}_{\mathrm{P}}(n) \leq Q(n)$. Consequently, The result holds by Definition 5. □

Theorem 6. *For every function* f *in* FBQP *with polynomial bound* $Q \in \mathbb{N}[X]$*, there is a* PFOQ *program* P *such that* $[\![\mathrm{P}]\!] \circ \phi_Q$ *computes* f *with probability* $\frac{2}{3}$*.*

Proof. Consider a function f in FBQP with polynomial bound Q and an input $|\psi\rangle \in \mathcal{H}$. By Theorem 4 there exists a function F in $\widehat{\square_1^{\mathrm{QP}}}$ such that $F \circ \phi_Q$ computes f with probability $\frac{2}{3}$. By Theorem 5, there exists a program P_F such that $[\![\mathrm{P}_F]\!] = F$. Finally, $[\![\mathrm{P}_F]\!] \circ \phi_Q$ computes f with probability $\frac{2}{3}$. □

Lemma 4 *For any circuit* $C_n \triangleq \circ_{i=1}^{k} U(cs_i, l_i)$*, with a unitary gate* U*, pairwise orthogonal* $cs_1, \ldots, cs_k \in \mathrm{Cst}$*, and* $l_1, \ldots l_k \in \mathcal{L}(\mathbb{N})$*, there exists a circuit* C *using one controlled gate* U*,* $O(kn)$ *gates, and* $O(k)$ *ancillas, and such that* $[\![C]\!] = [\![C_n]\!]$*.*

Proof. We describe the circuit that achieves the result and then prove its correctness. Let a_i, for $i \in \{1,\ldots,k\}$, be ancillas in the zero state. Define $C \triangleq C^1 \circ G(\cdot[a_k = 1], l_k) \circ C^2$ with

$$
C^1 \triangleq \circ_{i=1}^{k} NOT(cs_i, a_i) \circ (\circ_{i=1}^{k-1} NOT(\cdot[a_i = 1], a_k)) \circ (\circ_{i=1}^{k-1} \mathrm{SWAP}(\cdot[a_i = 1], l_i, l_k))
$$

and C^2 defined from C^1 by reversing the order of the gates. The total number of ancillas is k. Circuits C^1 and C^2 contain a constant number of controlled NOT gates and, in the worst case, $O(kn)$ controlled $SWAP$ gates (depending on the overlap between l_k and each l_i).

Consider a computational basis state $|x\rangle$, for $x \in \{0,1\}^n$. Since all control structures cs_i are pairwise orthogonal, after the two compositions of NOT circuits in C^1, $|x\rangle$ satisfies $\cdot[a_i := 1]$ iff it satisfies cs_i, for $i \in \{1,\ldots,k-1\}$. Moreover, $|x\rangle$ satisfies $\cdot[a_k := 1]$ iff there exists i such that it satisfies cs_i. Therefore, each $SWAP$ sends wires in l_i into l_k iff $|x\rangle$ satisfies cs_i and then U is applied on l_k if it satisfies one of the control structure cs_i. As circuit C^2 reverts the actions of C^1, afterwards, the qubits are in the right positions, all ancillas are set to zero, and the result on non-ancillary qubits is the same as in C_n. □

Lemma 5 *Orthogonality is an invariant property of the control structures in l_{Cst} of the subroutine* **optimize**. *In other words, for any two distinct pairs (cs, S), (cs', S') in l_{Cst}, cs and cs' are orthogonal.*

Proof. The proof is done by induction on a procedure statement, doing a case analysis on the rules of **optimize**. The initial value of l_{Cst} is of the form $[(cs, \mathrm{S}^{\mathrm{proc}})]$, which trivially satisfies the condition. Consider the possible cases for a pair (cs, S) in l_{Cst}. By induction hypothesis, assume that cs is orthogonal to all other control structures. For Composition, quantum case and classical **if**, the replacing pair is either of the shape (cs, S'), or it is two pairs with $cs[n := 0]$ and $cs[n := 1]$, therefore we conclude that the resulting l_{Cst} has pairwise orthogonal controls.

For a procedure call, we consider two cases. The first case occurs when a control statement $(\cdot[a := 1], \mathrm{S}^{\mathrm{proc}})$ is added to the list l_{Cst} and a is an ancilla controlled by cs. Since the original pair is removed, this construction also satisfies the invariant property. In the second case, there already exists an ancilla a assigned to the triple (proc, i, j). Let us argue by induction on the number of previous procedure call instances for this ancilla. Let $cs_1, \ldots, cs_k$ be control structures such that a computational basis state $|x\rangle$ satisfies $\cdot[a := 1]$ iff it satisfies one of the cs_i, which we assume by induction hypothesis are pairwise orthogonal. By the definition of PFOQ, cs will be orthogonal to each of the cs_i, and so after gate $NOT(cs, a)$, a state $|x\rangle$ will satisfy $\cdot[a := 1]$ iff it satisfies any of $cs, cs_1, \ldots, cs_k$. Since cs was orthogonal to all other control structures appearing in l_{Cst}, and the original pair is removed, the result l_{Cst} will have pairwise orthogonal controls. □

Theorem 7. *For any program* P *in* PFOQ, **compile**(P, n) *runs in time* $O(n^{2|\mathrm{P}|+1})$.

Proof. We first show that an execution of the optimize subroutine performs at most $O(n^2)$ calls to **compr**. We use a simple counting argument. The dictionary Anc ensures that ancillas related to the same $(\mathrm{proc}', \mathrm{i}, \mathrm{j}) \in \mathrm{Procedures} \times \mathbb{Z} \times \{1,\ldots,n\}$ will only be created once. The classical parameter can either be updated to a new constant or by adding (or subtracting) a constant and this can be done at most $O(n)$ times as procedure calls also remove at least an element

from the sorted set parameter. So the range of values for i is $O(n)$, therefore the space of possible values has size at most $O(|\mathrm{P}| \cdot n^2)$. Hence $O(n^2)$ ancillas can be created, which also bounds the number of calls to **compr**.

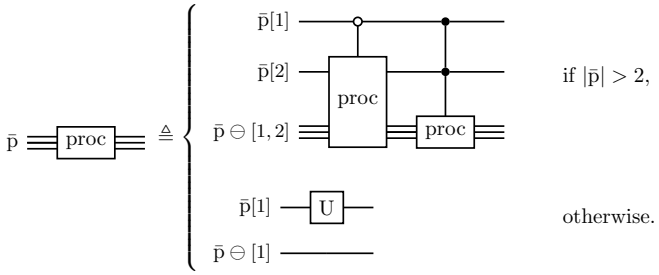
Second, consider an execution of **compr**. Subroutine **compr** calls itself directly only outside of recursive procedure calls, meaning that these cases consist of constant time circuit constructions. On the case of a call to a recursive procedure, it does a single call to **optimize** which creates at most $O(n^2)$ calls back to **compr**. Each of these instances of **compr** either does a non-recursive circuit construction of constant size, or does a call to **optimize** for a procedure of strictly lower rank, as defined in the proof of Lemma 2. Since there are at most $O(n^2)$ such calls, the maximum total number of procedure calls is $O(n^{2\,rk(\mathrm{P})}) \in O(n^{2|\mathrm{P}|})$ (see proof of Lemma 2 for a definition of rank rk). Each of these recursive procedure calls can have at most a constant number of merge constructions with linear complexity, therefore the total number of gates is $O(n^{2|\mathrm{P}|+1})$. □

B Example of circuit compilation with function merging

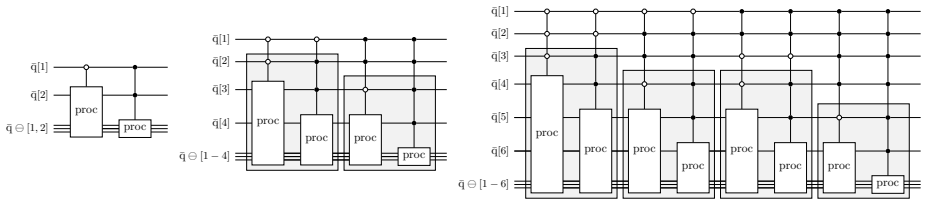
Consider the following program in PFOQ, defined for some unitary gate U:

```
P(q̄) ≜ decl proc(p̄){
         if |p̄| > 2 :
           qcase p̄[1] of {0 → call proc(p̄ ⊖ [1]); ,
                          1 → qcase p̄[2] of {0 → skip; ,
                                             1 → call proc(p̄ ⊖ [1,2]);
                               }
           }
         else p̄[1] *= U;
       }
       :: call proc(q̄);
```

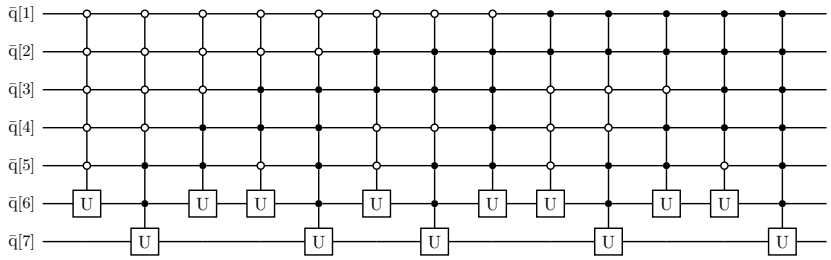
Consider first a naive approach to building a circuit representing this program. The circuit representation of P can be seen as:



For a large input length, creating the circuit by applying this definition would deal the following progression.

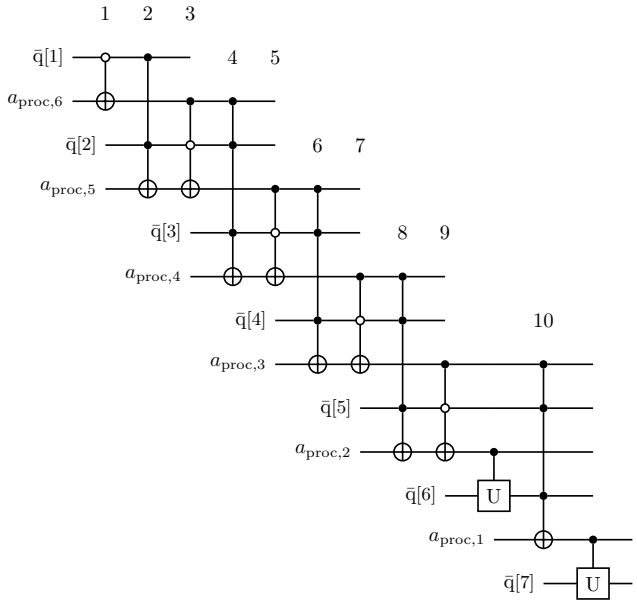


It is easy to see that this construction results in $O(|\bar{q}|2^{|\bar{q}|})$ gates. Consider the following example of input length $|\bar{q}| = 7$. Each Toffoli gate carries a complexity cost of $O(|\bar{q}|)$.



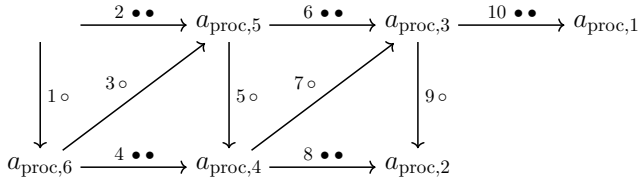
This is the scenario that **optimize** subroutine (Algorithm 2) avoids. By making use of the fact that there are multiple calls to proc with same input size with orthogonal controls, calls to sorted sets of equal size can be merged onto the same ancilla.

The following is the circuit that simulates P on the same input size, using 6 ancillas that, in this example, are not reset to zero for ease of reading. We have also indexed each Toffoli gate, which have $O(1)$ cost since they have constant size (at most 3).



In the specific case of procedure proc, there is no need to move around qubits while merging, which simplifies the construction and allows the circuit to scale with $O(|\bar{q}|)$. We represent each ancilla created for proc on input length i as $a_{\text{proc},i}$ (note that there is no integer parameter in this case so we omit it here in the index of ancillas). Since the only non-recursive cases are for input length $i = 1, 2$, those are the only ancillas with controlled gate U.

We can graphically see how the circuit is built linearly. The directed graph below shows the creation of each ancilla $a_{\text{proc},i}$, where each node is an ancilla and each edge indicates a (generalized) Toffoli gate, numbered exactly as in the circuit. We further add a tag $\circ$ or $\bullet\bullet$ to differentiate between the two types of Toffoli gate, which represent the two recursive branches, $\circ$ for the vertical and diagonal edges where the first qubit is in state 0, and $\bullet\bullet$ for the horizontal edges which refer to the case where the first two qubits are both in state 1.



Nodes have as incoming edges the branches that reach the same quantum input length and are merged in the same ancilla. If we were to represent the circuit for larger input lengths, we would obtain a continuation of this directed graph to the right, following the same pattern.

C Example: quantum teleportation

Quantum teleportation is an interesting technique that allows for transportation of qubits between far away agents, Alice and Bob. If Alice and Bob share n EPR states, they can teleport any n-length state between their labs.

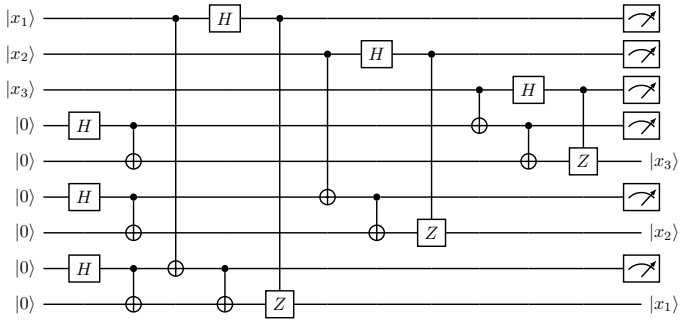
In our case, we consider an input state of length n, and we extend it with qubits $|0\rangle^{\otimes 2n}$. The following is a FOQ program where we use the $2n$ zero state qubits to create a Bell state and then perform the teleportation for each qubit, such that the i-th qubit of the initial state appears in position $3n - 2(i-1)$ of the final state for $1 \leq i \leq n$.

---

decl createBell($\bar{\mathrm{p}}$){
  if $|\bar{\mathrm{p}}| \geq 3$:
    $\bar{\mathrm{p}}[|\bar{\mathrm{p}}| - 1]$ ***=** H;
    CNOT($\bar{\mathrm{p}}[|\bar{\mathrm{p}}| - 1], \bar{\mathrm{p}}[|\bar{\mathrm{p}}|]$)
    call createBell($\bar{\mathrm{p}} \ominus [1, |\bar{\mathrm{p}}| - 1, |\bar{\mathrm{p}}|]$);
  else skip;},

decl teleport($\bar{\mathrm{p}}$){
  if $|\bar{\mathrm{p}}| \geq 3$:
    CNOT($\bar{\mathrm{p}}[1], \bar{\mathrm{p}}[|\bar{\mathrm{p}}| - 1]$)
    $\bar{\mathrm{p}}[1]$ ***=** H;
    CNOT($\bar{\mathrm{p}}[|\bar{\mathrm{p}}| - 1], \bar{\mathrm{p}}[|\bar{\mathrm{p}}|]$)
    qcase $\bar{\mathrm{p}}[1]$ **of** $\{0 \to$ **skip**;, $1 \to \bar{\mathrm{p}}[|\bar{\mathrm{p}}|]$ ***=** $\mathrm{Ph}^{\lambda x.\pi}(0);\}$
    call teleport($\bar{\mathrm{p}} \ominus [1, |\bar{\mathrm{p}}| - 1, |\bar{\mathrm{p}}|]$);
  else skip;}
::
call createBell($\bar{\mathrm{q}}$);
call teleport($\bar{\mathrm{q}}$);

It can be easily shown that the program is in PFOQ. We apply the principle of deferred measurement to avoid performing any measurements until the end, and therefore use typical controlled NOT and Z gates as opposed to the same gates controlled on results of measurements. The following is an example of the circuit compilation for the teleportation of a state of length $n = 3$.


$|x_1\rangle$
$|x_2\rangle$
$|x_3\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
H
H
H
H
H
Z
Z
Z
$|x_3\rangle$
$|x_2\rangle$
$|x_1\rangle$

D Example of derivation tree

The derivation tree π described below corresponds to the evaluation of procedure rot of Example 1 on an input of length 2.

Let $|\psi'\rangle \triangleq |0\rangle\!\rangle\langle\!\langle 0|_2 |\psi\rangle + |1\rangle\!\rangle\langle\!\langle 1|_2 \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^{2-1}} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} |\psi\rangle$

in:

Let

$$\pi_1 \triangleq \left\{ \dfrac{\dfrac{\overline{(\bar{\mathrm{p}}, [1,2]) \Downarrow_{\mathcal{L}(\mathbb{N})} [1,2]} \quad \overline{(2, [1,2]) \Downarrow_{\mathbb{Z}} 2}}{(\bar{\mathrm{p}}[2], [1,2]) \Downarrow_{\mathbb{N}} 2} \quad \dfrac{}{(\mathbf{skip}; , |\psi\rangle, \{1\}, [1,2]) \xrightarrow{0} (\top, |\psi\rangle, \{1\}, [1,2])} \quad \dfrac{\dfrac{\overline{(\bar{\mathrm{p}}, [1,2]) \Downarrow_{\mathcal{L}(\mathbb{N})} [1,2]} \quad \overline{(1, [1,2]) \Downarrow_{\mathbb{Z}} 1}}{(\bar{\mathrm{p}}[1], [1,2]) \Downarrow_{\mathbb{N}} 1} \quad \dfrac{\overline{(2, [1,2]) \Downarrow_{\mathbb{Z}} 2}}{(\mathrm{Ph}^{\lambda y.\pi/2^{y-1}}(2), [1,2]) \Downarrow_{\mathbb{C}^{2\times 2}} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^{2-1}} \end{pmatrix}}}{(\bar{\mathrm{p}}[1] \mathbin{*\!\!=} \mathrm{Ph}^{\lambda y.\pi/2^{y-1}}(2); , |\psi\rangle, \{1\}, [1,2]) \xrightarrow{0} (\top, \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^{2-1}} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} |\psi\rangle, \{1\}, [1,2])} }{(\mathbf{qcase}\ \bar{\mathrm{p}}[2]\ \mathbf{of}\ \{0 \to \mathbf{skip}; , 1 \to \bar{\mathrm{p}}[1] \mathbin{*\!\!=} \mathrm{Ph}^{\lambda y.\pi/2^{y-1}}(2); \}, |\psi\rangle, \{1,2\}, [1,2]) \xrightarrow{0} (\top, |\psi'\rangle, \{1,2\}, [1,2])} \right.$$

and

$$\pi_2 \triangleq \left\{ \dfrac{\dfrac{\overline{(\bar{\mathrm{p}}, [1,2]) \Downarrow_{\mathcal{L}(\mathbb{N})} [1,2]} \quad \overline{(2, [1,2]) \Downarrow_{\mathbb{Z}} 2}}{(\bar{\mathrm{p}} \ominus [2], [1,2]) \Downarrow_{\mathcal{L}(\mathbb{N})} [1]} \quad \dfrac{\overline{(2, [1,2]) \Downarrow_{\mathbb{Z}} 2} \quad \overline{(1, [1,2]) \Downarrow_{\mathbb{Z}} 1}}{(2+1, [1,2]) \Downarrow_{\mathbb{Z}} 3} \quad \dfrac{\dfrac{\overline{(\bar{\mathrm{p}}, [1]) \Downarrow_{\mathcal{L}(\mathbb{N})} [1]}}{(|\bar{\mathrm{p}}|, [1]) \Downarrow_{\mathbb{Z}} 1} \quad \overline{(1, [1]) \Downarrow_{\mathbb{Z}} 1}}{(|\bar{\mathrm{p}}| > 1, [1]) \Downarrow_{\mathbb{B}} \mathbf{false}} \quad \dfrac{}{(\mathbf{skip}; , |\psi'\rangle, \{1,2\}, [1]) \xrightarrow{0} (\top, |\psi'\rangle, \{1,2\}, [1])}}{(\mathbf{if}\ |\bar{\mathrm{p}}| > 1 \ldots, |\psi'\rangle, \{1,2\}, [1]) \xrightarrow{0} (\top, |\psi'\rangle, \{1,2\}, [1])} }{(\mathbf{call}\ \mathrm{rot}[2+1](\bar{\mathrm{p}} \ominus [2]); , |\psi'\rangle, \{1,2\}, [1,2]) \xrightarrow{1} (\top, |\psi'\rangle, \{1,2\}, [1,2])} \right.$$

in:

Let $\mathrm{S} \triangleq \mathbf{if}\ |\bar{\mathrm{p}}| > 1\ \mathbf{then}\ \mathbf{qcase}\ \bar{\mathrm{p}}[2]\ \mathbf{of}\ \{0 \to \mathbf{skip}; , 1 \to \bar{\mathrm{p}}[1] \mathbin{*\!\!=} \mathrm{Ph}^{\lambda y.\pi/2^{y-1}}(2); \}\ \mathbf{call}\ \mathrm{rot}[2+1](\bar{\mathrm{p}} \ominus [2]);\ \mathbf{else}\ \mathbf{skip};$

and $\mathrm{S}' \triangleq \mathbf{qcase}\ \bar{\mathrm{p}}[2]\ \mathbf{of}\ \{0 \to \mathbf{skip}; , 1 \to \bar{\mathrm{p}}[1] \mathbin{*\!\!=} \mathrm{Ph}^{\lambda y.\pi/2^{y-1}}(2); \}\ \mathbf{call}\ \mathrm{rot}[2+1](\bar{\mathrm{p}} \ominus [2]);$

in:

$$\pi \triangleq \left\{ \dfrac{\overline{(\bar{\mathrm{p}}, [1,2]) \Downarrow_{\mathcal{L}(\mathbb{N})} [1,2]} \quad \overline{(2, [1,2]) \Downarrow_{\mathbb{Z}} 2} \quad \dfrac{\dfrac{\dfrac{\overline{(\bar{\mathrm{p}}, [1,2]) \Downarrow_{\mathcal{L}(\mathbb{N})} [1,2]}}{(|\bar{\mathrm{p}}|, [1,2]) \Downarrow_{\mathbb{Z}} 2} \quad \overline{(1, [1,2]) \downarrow_{\mathbb{Z}} 1}}{(|\bar{\mathrm{p}}| > 1, [1,2]) \Downarrow_{\mathbb{B}} \mathbf{true}} \quad \dfrac{\pi_1 \qquad \pi_2}{(\mathrm{S}', |\psi\rangle, \{1,2\}, [1,2]) \xrightarrow{1+0} (\top, |\psi'\rangle, \{1,2\}, [1,2])}}{(\mathrm{S}, |\psi\rangle, \{1,2\}, [1,2]) \xrightarrow{1} (\top, |\psi'\rangle, \{1,2\}, [1,2])}}{(\mathbf{call}\ \mathrm{rot}[2](\bar{\mathrm{p}}); , |\psi\rangle, \{1,2\}, [1,2]) \xrightarrow{1+1} (\top, |\psi'\rangle, \{1,2\}, [1,2])} \right.$$