Quantum Weakest Preconditions for Reasoning about Expected Runtimes of Quantum Programs

Junyi Liu
Institute of Software, CAS
Beijing, China
liujy@ios.ac.cn

Gilles Barthe
Max Planck Institute for Security and Privacy (MPI-SP)
Bochum, Germany
gjbarthe@gmail.com

Li Zhou
Max Planck Institute for Security and Privacy (MPI-SP)
Bochum, Germany
zhou31416@gmail.com

Mingsheng Ying
Institute of Software, CAS
Beijing, China
yingms@ios.ac.cn

ABSTRACT
We study expected runtimes for quantum programs. Inspired by recent work on probabilistic programs, we first define expected runtime as a generalisation of quantum weakest precondition. Then, we show that the expected runtime of a quantum program can be represented as the expectation of an observable (in physics). A method for computing the expected runtimes of quantum programs in finite-dimensional state spaces is developed. Several examples are provided as applications of this method, including computing the expected runtime of quantum Bernoulli Factory – a quantum algorithm for generating random numbers. In particular, using our new method, an open problem of computing the expected runtime of quantum random walks introduced by Ambainis et al. (STOC 2001) is solved.

CCS CONCEPTS
• Theory of computation → Pre- and post-conditions; Program analysis; Denotational semantics.

KEYWORDS
Quantum programming, quantum weakest precondition, expected runtime, physical observable, termination, quantum random walk

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
LICS ’22, August 2–5, 2022, Haifa, Israel.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9551-5/22/08 ... $15.00
https://doi.org/10.1145/3531130.3533327

1 INTRODUCTION
Over the last few years, one has seen exciting progress in quantum computing hardware, such as Google’s 53-qubit Sycamore [5], IBM Q [13] and USTC’s 62-qubit Zu Chongzhi [16]. This progress further motivates one to expect that the Noisy Intermediate-Scale Quantum (NISQ) technology [32] will find some practical applications in the next 5-10 years.

Quantum Resource and Runtime Estimation: Resource and runtime estimation will be particularly important in programming NISQ devices because they are too resource-constrained to support error correction, and running long sequences of operations on them is impractical. On the other hand, resource and runtime estimation may help us understand the separation between the quantum algorithms that can be run on NISQ devices and those that must wait for a larger quantum computer. Indeed, quantum resource and runtime estimation problem has already attracted the attention of quantum computing researchers; for example, resource and timing analysis was incorporated into quantum compilation framework ScaffCC [20]; an estimation of required qubits and quantum gates for Shor’s algorithm to attack ECC (Elliptic Curve Cryptography) was given in [33]. But current research in this area has been carried out mainly in a manner of case by case. Certainly, a more principled approach to this problem would be desirable.

Techniques in Probabilistic Programming: Recently, a series of powerful techniques for resource and runtime estimation of probabilistic programs have been proposed (see for example [6, 7, 9, 11, 12, 25, 31]). In particular, inspired by Nielson’s Hoare-like proof system for reasoning about the running times of non-probabilistic programs [29], a weakest precondition calculus was developed in [21, 22] for analysing the expected runtimes of randomised algorithms and probabilistic programs. It has been successfully applied to estimate the expected runtimes of several interesting example probabilistic programs, including the coupon collector’s problem, one-dimensional random walk and randomised binary search. Furthermore, an analysis that can derive symbolic bound on the expected resource consumption of probabilistic programs was presented in [26] by effectively combining the weakest
precondition reasoning for probabilistic programs \cite{10, 21, 27, 37} with the automatic amortised resource analysis (AARA) for non-probabilistic programs \cite{8, 9, 17, 18}. The strength of this approach is that it can be fully automated by reducing the bound analysis to LP (Linear Program) solving, and its effectiveness was demonstrated by automatic analysis of a large number of challenging randomised algorithms and probabilistic programs.

From Quantum Weakest Preconditions to Quantum Expected Runtimes: The well-known statistical nature of quantum systems immediately suggests the possibility of extending the techniques discussed above for solving the corresponding problems in quantum computing. Fortunately, a foundation for this research was already laid in the seminal work \cite{15} where the notion of weakest quantum precondition was defined. The aim of this paper is to extend the line of research initiated by \cite{15} and to develop a weakest precondition calculus for reasoning about the expected runtimes of quantum programs. Our basic ideas in the extension from quantum weakest precondition to quantum expected runtimes are largely inspired by the results achieved in the studies of expected runtimes of probabilistic programs \cite{21, 22}. But several challenges exist in the transition from the probabilistic case to the quantum case:

\begin{itemize}
  \item **Conceptual challenge:** The expected runtime \(ert(S)\) of a probabilistic program \(S\) is defined in \cite{21, 22} as a transformer of runtime functions, which are modelled as mappings from the state space to nonnegative real numbers or \(\infty\) that are linear on the probabilistic combination of states. Whenever generalised to the quantum case, expected runtime functions are no longer linear on the superposition of pure states due to the Born rule in quantum mechanics. Thus, we need to find an appropriate interpretation for them as observables in quantum physics (or mathematically as Hermitian operators) so that they can be effectively manipulated and computed. We resolve this issue using several mathematical techniques developed in the previous work on quantum weakest preconditions \cite{15} and quantum Hoare logic \cite{38}.
  \item **Computational challenge:** Although quantum gates are modelled as unitary operators, quantum measurements are used in the guards of conditional statements and loops. Thus, super-operators are inevitably involved in their denotational semantics and computing their weakest preconditions. However, super-operators are (completely positive) mappings from (linear) operators to themselves and are much harder to manipulate than ordinary operators. Fortunately, some matrix representations of super-operators in Hilbert spaces can be developed based on Theorem 1. More explicitly, by combining the observable representation with the matrix representation of super-operators, we develop a method for computing the expected runtimes of quantum programs. This method works both numerically and symbolically, with the numerical computation fully automated. It is worth mentioning that our method can deal with the case of infinite execution paths, which was excluded in the method of \cite{20}.
  \item The effectiveness of our method is tested by several case studies, including a key step of quantum Bernoulli factory for random number generation. In particular, we solve an open problem of computing the expected runtime of the quantum walk on an \(n\)-circle for any \(n\) and an arbitrary initial state. The previously known result about this problem is that the expected runtime is \(n\) when starting in a basis state for \(n < 30\) (see \cite{40} or Section 5.1.3 of \cite{39}).
\end{itemize}

Related Works: The research on formal reasoning about expected runtimes of quantum programs was initiated in \cite{30}. The aim of \cite{30} is the same as that of this paper, but the technical results of the two papers are very different. After giving a weakest precondition-style definition of expected runtime for quantum programs, \cite{30} focused on a case study of analyzing the runtime of (a simplified version of) the BB84 quantum key distribution protocol, and our contributions 2) - 4) listed above were not considered there. Perhaps, a more essential difference between \cite{30} and our work is that in \cite{30}, quantum programs are treated as a kind of probabilistic programs and their expected runtimes are evaluated by directly applying the techniques for probabilistic programs. We argue that such a generalization does not make use of the unique properties of quantum systems, and thus cannot be used to handle programs that have quantum-specific behaviours. To be more specific, let us consider the example quantum walk, where quantum interference happens. Using the definition of \cite{30}, a fixed point equation can be derived whose solutions are upper bounds of the expected runtime of the quantum walk, but solving this equation is hard because the quantum walk is treated as a probabilistic program and thus its quantum-specific feature is lost. In contrast, in our approach, a different fixed point equation is derived, and it can be solved by sufficiently exploiting mathematical properties of the involved quantum operations (as super-operators).

Organization of the Paper: We start from several working examples in Section 2.1, and then the syntax and semantics of quantum programs are reviewed in Section 2.2. The expected runtime
of quantum programs are defined in Section 3. The observable representation is presented in Section 4.1. Based on it, we prove the equivalence of almost surely termination and positive almost-sure termination of quantum programs in Section 4.2. A method for computing the expected runtimes of quantum programs is presented in Section 5. The case studies are given in Section 6. For the sake of readability, proofs of our results can be found in the extended version of this work [24].

2 QUANTUM PROGRAMS

In this section, we set our stage by reviewing the syntax and semantics of the quantum programs considered in this paper. We assume that the reader is familiar with basic ideas of quantum computing; otherwise, she/he can consult the standard textbook [28] or the preliminary sections of quantum programming literature, e.g. [34, 35].

2.1 Working Examples

Let us start with several simple examples. They are deliberately chosen as the quantum analogues of some examples considered in [21, 22] so that the reader can observe the similarity and subtle differences between probabilistic and quantum programs.

We first consider the process that keeps tossing a fair coin until the first head occurs. This process can be described as a probabilistic program:

$$ P_{geo} = \text{while } (c = 1) \{ c : = 0 \text{ if } \frac{1}{2} < c < 1 \} $$

where 0, 1 are used to indicate the heads and tails, respectively, and $P_1$ and $P_2$ stands for a probabilistic choice that chooses to execute $P_1$ with probability $a$ and to execute $P_2$ with probability $1 - a$. A quantum analogue of this program is given as the following:

Example 2.1 (Quantum Geometric Distribution). A quantum coin can be modelled as a quantum bit (qubit for short) variable $q$. Using the Dirac notation, we write $|1\rangle, |0\rangle$ for the head and tail, respectively. Quantum coin $q$ can be in states $|1\rangle, |0\rangle$ as a classical coin, but it can also be in a superposition $|a\rangle + \langle b|1\rangle$, where $a, b$ are complex numbers satisfying the normalisation condition $|a|^2 + |b|^2 = 1$. Thus, its state space is the 2-dimensional Hilbert space $\mathcal{H}_0 = \text{span}(|0\rangle, |1\rangle)$ with head $|1\rangle$ and tail $|0\rangle$ as its basis states. To detect the state of the quantum coin, we need to perform quantum measurement on it. Here, we use the measurement $M$ in the computational basis $\{|0\rangle, |1\rangle\}$, which is mathematically modelled as a pair of operators $M = (M_0, M_1)$ with $M_0 = |0\rangle \langle 0|, M_1 = |1\rangle \langle 1|$. For example, when performing $M$ on a quantum coin in superposition $|a\rangle + \langle b|1\rangle$, we will see the tail $|0\rangle$ with probability $|a|^2$ and the head $|1\rangle$ with probability $|b|^2$.

A quantum program that behaves in a way similar to $P_{geo}$ is defined as:

$$ Q_{geo} = \text{while } M[q] = 1 \text{ do } q := H[q] \text{ od} $$

where $H$ is the Hadamard gate represented by the matrix

$$ H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} $$

which transforms the tail $|0\rangle$ and head $|1\rangle$ into their equal superpositions $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

It is interesting to compare programs $P_{geo}$ and $Q_{geo}$ carefully. First, coin tossing is treated in $P_{geo}$ as an abstract program construct $\oplus$ without specifying how to implement it. However, in $Q_{geo}$ we have to explicitly describe it as a physical process: it is realised by first applying the Hadamard gate $H$ on quantum coin $q$ and then measuring $q$ in the computational basis. Second, the coin $c$ in $P_{geo}$ is always in either state 0 (tail) or 1 (head) although we cannot predict its next state with certainty before tossing it. But if the quantum coin $q$ is initialised in tail $|0\rangle$ (respectively, head $|1\rangle$), then the Hadamard gate will transform it into a superposition $|+\rangle$ (or $|-\rangle$) of the head and tail. Third, checking the loop guard in $P_{geo}$ does not change the state of coin $c$. However, in $Q_{geo}$ we need to perform measurement $M$ on quantum coin $q$ when checking the loop guard in order to acquire information about $q$ and the measurement will change the state of $q$: for example, if $q$ is in state $|+\rangle$ before the measurement, then after the measurement $q$ will be in state $|0\rangle$ with probability $\frac{1}{2}$ and in $|1\rangle$ with probability $\frac{1}{2}$, but no longer in $|+\rangle$. It is even more interesting to note that there are (uncountably) infinitely many operators and measurements rather than $H$ and $M$ in $Q_{geo}$ suited to implement the fair coin tossing.

Another example considered in [21] is the following one-dimensional random walk with an absorbing boundary:

$$ C_{rw} \equiv \text{if } x = 10 \text{ then } (x : = x - 1 \text{ if } x < 10 \text{ else } x : = x + 1) $$

It starts from position 10 and shifts on a line to the left or right with equal probability in each step until reaching the boundary at position 0. A quantum counterpart of this random walk, namely the semi-infinite Hadamard walk, was defined in [4].

Example 2.2 (Hadamard Walk). Let $q$ be a quantum coin with a 2-dimensional state space $\mathcal{H}_q$ spanned by orthogonal basis states $|L\rangle$ and $|R\rangle$, indicating directions Left and Right, respectively. We introduce a quantum variable $p$ with state space $\mathcal{H}_p$ spanned by orthogonal basis $|n\rangle : n \in \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers, and $|n\rangle$ is used to denote position $n$ on a line. Then the Hadamard walk is considered as a composite system of $q$ and $p$ and thus its state space is the tensor product $\mathcal{H} = \mathcal{H}_q \otimes \mathcal{H}_p$. A measurement that determines whether the system is at position 0 is described as $N = \{N_0, N_1\}$ where $N_0 = |0\rangle \langle p|0\rangle$ and $N_1 = |p\rangle - |0\rangle$, and $I_p$ is the identity operators on $\mathcal{H}_p$. To describe one step of the walk, we define a shift operator $S$ by

$$ S(L, n) = |L, n - 1\rangle, \quad S(R, n) = |R, n + 1\rangle, $$

(together with linearity), meaning that the position shifts one position to the left or to the right according to the state $|L\rangle$ or $|R\rangle$ of coin $q$.

Intuitively, the quantum walk repeatedly behaves as follows:

(1) Perform measurement $N$ to see whether the system is at position 0.

(2) If it is at 0, then terminate; otherwise, apply Hadamard gate $H$ to quantum coin $q$ in order to generate an equal superposition of the directions Left and Right, and then move the position according to shift operation $S$ and goto step 1.

Formally, it can be written as a quantum program:

$$ Q_{rw} \equiv \text{if } N[q] = 1 \text{ do } q := H[q] \text{; } q, p := S[q, p] \text{ od} $$

An apparent difference between $C_{rw}$ and $Q_{rw}$ is that the latter can move to the left and right simultaneously; for example, if currently the coin is in state $|R\rangle$ and the position is $n$, then applying
Hadamard gate $H$ yields a superposition $\frac{1}{\sqrt{2}}(|L\rangle - |R\rangle)$ of directions $L$ and $R$, and shift operator $S$ transforms to system to state $\frac{1}{\sqrt{2}}(|L, n - 1\rangle - |R, n + 1\rangle)$. A more essential difference between $C_{rw}$ and $Q_{rw}$ is the so-called quantum interference: the coefficients (called probability amplitudes) in a quantum state can be negative (and imaginary) numbers (see the above example states), and thus two paths of the walk $Q_{rw}$ with positive and negative amplitudes respectively can cancel one another. This feature has been extensively exploited to design quantum walks-based algorithms faster than the corresponding classical algorithms (see for example, [1–3, 36]).

The following probabilistic program was used in [21] to show a fundamental difference between the runtime of non-probabilistic programs and that of probabilistic programs:

$$C \equiv C_1 : x := 1; c := 1;$$
$$\text{while } (c = 1) \{ c := 0 @ c \}; e := 1; x := 2x \}$$
$$C_2 : \text{while } (x > 0) \{ x := x - 1 \}$$

Obviously, both of subprograms $C_1$ and $C_2$ have a finite expected runtime on all inputs. However, it is easy to see that $C_1; C_2$ has an infinite expected runtime. A quantum variant of this program is given as the following:

**Example 2.3.** Let quantum variables $q, p$, Hilbert space $\mathcal{H}_q, \mathcal{H}_\omega$, and measurement $N$ be the same as in Example 2.2. We introduce the following two unitary operators on $\mathcal{H}_\omega$, as quantum mimics of assignments $x := x - 1$ and $x := 2x$, respectively, in the above program $C$:

- The left-shift operator $T_L$ is defined by $T_L|n\rangle = |n - 1\rangle$ for every $n \in \mathbb{Z}$.
- The duplication operator $D$ is defined as follows: $D|n\rangle = |2n\rangle$ for $n \geq 0$ and $D|n\rangle = |K(n)\rangle$ for $n < 0$. To make $D$ be a unitary operator on $\mathcal{H}_\omega$, $K$ must be chosen as a one-to-one mapping:
  $$\{-n : n \geq 1\} \rightarrow \{-n : n \geq 1\} \cup \{2n - 1 : n \geq 1\},$$
  e.g. $K(1 - 2n) = 2n - 1$ and $K(-2n) = -n$ for $n \geq 1$.

Then we can define quantum programs:

$$Q \equiv Q_1 : \text{while } M[q] = 1 \text{ do } p := H[q] \text{; } p := D[p] \text{ od};$$
$$Q_2 : \text{while } N[p] = 1 \text{ do } p := T_L[p] \text{ od} \quad (6)$$

The above quantum program $Q$ behaves similarly to probabilistic program $C$ when the input state is $|R, 1\rangle$. It is worth noting an interesting difference between the left-shift operator $T_L$ in $Q$ and the shift operator $S$ in Example 2.2. Since $T_L$ always moves in the same left direction, it can be defined simply on $\mathcal{H}_\omega$. In contrast, $S$ moves in the direction determined by the quantum coin $q$, so it has to be defined on $\mathcal{H}_q \otimes \mathcal{H}_\omega$.

The three examples presented above may give the reader the impression that quantum programs are very similar to probabilistic programs. But we must point out that quantum programs are notoriously harder to analyze than probabilistic ones, as we will see from our example of the quantum walk in Section 6.2.

### 2.2 Syntax

The examples presented in the above subsection should give the reader intuition about basic quantum program constructs. In this subsection, we formally define the syntax of quantum programs studied in this paper.

We choose to use the quantum $\text{while}$-language defined in [38, 39] for a consistency with [21, 22], where probabilistic $\text{while}$-language was employed. We assume a countably infinite set $\text{qVar}$ of quantum variables and use $q, q_0, q_1, q_2, \ldots$ to denote them. The state Hilbert space of a quantum variable $q$ is denoted $H_q$. A quantum register is a finite sequence of distinct quantum variables. The state space of a quantum register $\overline{q} = q_0 \ldots q_n$ is then the tensor product $H_{\overline{q}} = \otimes_{i=0}^n H_{q_i}$.

**Definition 2.1 (Syntax [38]).** The set $\text{qProgs}$ of quantum while-programs is defined by the following syntax:

$$S ::= \text{skip} \mid S_1; S_2 \mid q ::= |0\rangle \mid \overline{q} : = U[\overline{q}] \quad (7)$$
$$\mid \text{if } (\exists m \cdot M[\overline{q}] = m \rightarrow S_m) \text{ fi} \quad (8)$$
$$\mid \text{while } M[\overline{q}] = 1 \text{ do } S \text{ od} \quad (9)$$

A brief explanation of the above program constructs is given as follows. The constructs $\text{skip}$ and sequential composition $S_1; S_2$ are similar to their counterparts in the classical or probabilistic $\text{while}$-programs. The initialisation $q ::= |0\rangle$ sets the quantum register $q$ to the basis state $|0\rangle$. The statement $\overline{q} : = U[\overline{q}]$ means that unitary transformation $U$ is performed on the quantum register $\overline{q}$. The construct in (8) is a quantum generalisation of the classical case statement. In the execution, measurement $M = \{M_m\}$ is performed on $\overline{q}$, and then a subprogram $S_m$ will be selected according to the measurement outcome. The statement in (9) is a quantum generalisation of $\text{while}$-loop, where the measurement $M$ has only two possible outcomes: if the outcome is 0, the program terminates, and if the outcome 1 occurs, the program executes the loop body $S$ and then continues the loop. Most of these constructs were already used in our working Examples 2.1, 2.2 and 2.3.

### 2.3 Semantics

For each quantum program $S$, we write $\text{var}(S)$ for the set of all variables $q \in \text{qVar}$ appearing in $S$. The Hilbert space of program $S$ is the tensor product $H_S = \otimes_{q \in \text{var}(S)} H_q$. Let $\mathcal{D}(H_S)$ be the set of all partial density operators (i.e. positive operators with the trace $\leq 1$) on $H_S$. A state of program $S$ is then represented by a partial density operator $\rho \in \mathcal{D}(H_S)$. Furthermore, a configuration is defined as a pair $(S, \rho)$ of a program $S$ and a state $\rho$. The operational semantics of quantum programs can be defined as a transition relation between configurations. Based on it, the following denotational semantics can be derived, and will be extensively used in this paper:

**Lemma 1 (Structural Representation of Denotational Semantics [38]).** For any input state $\rho \in H_S$, we have:

1. $[[\text{skip}]](\rho) = \rho$;
2. $[[q ::= |0\rangle]](\rho) = \sum_n |0\rangle_q(n|n\rangle_q|0\rangle)$;
3. $[[\overline{q} : = U[\overline{q}]](\rho) = U \rho U^\dagger]$;
4. $[[S_1; S_2]](\rho) = [[S_2]]([[S_1]](\rho))$;
5. $[[\text{if } (\exists m \cdot M[\overline{q}] = m \rightarrow S_m) \text{ fi}]](\rho) = \sum_m [[S_m]](M_m \rho M_m^\dagger)$;
6. for loop $[[\text{while } M[S] \equiv \text{while } M[\overline{q}] = 1 \text{ do } S \text{ od}]](\rho)_{k=0} \text{ is defined}$. 

$$[[\text{while } M[S]](\rho)_{k=0} = \left[\frac{[[\text{while } M[S]](\rho)]}{k=0}\right].$$
where \( \text{while}^{(k)}[M,S] \) is the \( k \)-fold iteration of the loop while:

\[
\text{while}^{(0)}[M,S] \equiv \text{abort},
\]

\[
\text{while}^{(k+1)}[M,S] = \begin{cases} \text{if } M[\overline{q}] = 0 \rightarrow \text{skip} \\ 1 \rightarrow S; \text{while}^{(k)}[M,S] \end{cases}
\]

for \( k \geq 0 \), \( [ \cdot ] \) stands for the least upper bound in the CPO of partial density operators with the Löwner order \( \subseteq \) (see [39], Lemma 3.3.2), and \text{abort} is a program that never terminates so that \( [\text{abort}] \)(\( \rho \)) = 0 for all \( \rho \).

We remark that a partial density operator \( \rho \in \mathcal{H}_S \) can be thought of as a representation of a sub-distribution of pure states in \( \mathcal{H}_S \). Thus, the sum of partial density operators in the above lemma is analogous to the sum of sub-distributions of states in probabilistic programming.

It immediately follows from the above lemma that the denotational semantics \( [S] \) of a quantum program is a (completely positive) super-operator, which is the mathematical formalism of quantum operations or the (discrete-time) dynamics of open quantum systems.

### 3.1 Definition

We only consider the case where the state Hilbert spaces of all quantum variables in a quantum program \( S \) are finite-dimensional, and thus \( \mathcal{H}_S \) is finite-dimensional too.

To simplify the presentation, let us first introduce several notations. For the measurement \( M = \{M_m\} \) in \text{if} statement (8), and for each possible measurement outcome \( m \), we define a super-operator \( E_{M_m}(\rho) = M_m\rho M_m^\dagger \) for all density operators \( \rho \). Similarly, for the measurement \( M = \{M_0, M_1\} \) in \text{while}-loop (9), we define super-operators \( E_0, E_1 \) as

\[
E_0(\rho) = M_0\rho M_0^\dagger, \quad E_1(\rho) = M_1\rho M_1^\dagger
\]

for all density operators \( \rho \).

A straightforward generalisation of the expected runtimes of probabilistic programs defined in [21, 22] yields the following:

**Definition 3.1.** The expected runtime of a quantum program \( S \) is a real-valued function \( \text{ERT}[S] : \mathcal{D}(\mathcal{H}_S) \rightarrow \mathbb{R} \cup \{\infty\} \) defined as follows:

1. \( \text{ERT}[\text{skip}](\rho) = 0 \);
2. \( \text{ERT}[q := [0]](\rho) = \text{tr} \rho \);
3. \( \text{ERT}[\overline{q} := U[\overline{q}]](\rho) = \text{tr} \rho \);
4. \( \text{ERT}[S_1; S_2](\rho) = \text{ERT}(S_1)(\rho) + \text{ERT}(S_2)([S_1]\)(\rho)) \);
5. \( \text{ERT}[\text{if}(m \cdot \overline{M}[\overline{q}] = m \rightarrow S_m)\](\rho) = \text{tr} \rho + \sum_m \text{ERT}[S_m](E_{M_m}(\rho)) \);
6. \( \text{ERT}[\text{while}^{[k]}[M,S]](\rho) = \lim_{k \rightarrow \infty} \text{ERT}[\text{while}^{[k]}[M,S]](\rho) \),

where \( \text{while}^{[k]}[M,S] \) is the first \( k \) iterations of the loop defined by:

\[
\text{while}^{[0]}[M,S] \equiv \text{skip},
\]

\[
\text{while}^{[k+1]}[M,S] = \begin{cases} \text{if } M[\overline{q}] = 0 \rightarrow \text{skip} \\ 1 \rightarrow S; \text{while}^{[k]}[M,S] \end{cases}
\]

for \( k \geq 0 \). [Note the difference between \text{while}^{[0]} defined here and \text{while}^{[0]} in Lemma 1. This makes \text{while}^{[k]} and \text{while}^{[k]} different for all \( k \geq 0 \).]

Intuitively, for any state \( \rho \in \mathcal{D}(\mathcal{H}_S) \), \( \text{ERT}[S]\)(\( \rho \)) is the expected runtime of program \( S \) on input \( \rho \). Our design decisions in the above definition are explained as follows:

- As usual, we choose to assume the expected runtime of \text{skip} to be zero. This will be used in defining the expected runtime of a \text{while}-loop. It should be noted that the runtime of \text{skip} is not equal to that of identity transformation \( q := I[q] \), which does nothing but takes 1 step.
- When applying to a density operator, the runtime of an initialisation or a unitary transformation is defined to be 1. We would like to extend the domain of \( \text{ERT}[S] \) from density operators to partial density operators for simplifying the presentation. It is reasonable to define their runtime applying to a partial density operator \( \rho \) as \( \text{tr} \rho \), which is the product of probability \( \text{tr} \rho \) and 1. Such a definition is consistent with the requirement that \( \text{ERT}[S] \) is linear.
- The runtime of sequential composition \( S_1; S_2 \) is defined in the same way as the case of probabilistic programs. It is worth noting that whenever \( \text{ERT}[S_1]\)(\( \rho \)) = \( \infty \), then the second summand \( \text{ERT}[S_2](\{[S_1]\})(\rho)) \) can be ignored.
- The runtime of an \text{if} statement is defined as the sum of the runtimes of all branches plus \( \text{tr} \rho \), which can be thought of as the time that the measurement in its guard takes.
- The runtime of a \text{while} statement is defined as the limit of the runtime of its unfolding (iterations).

The function \( \text{ERT} \) for probabilistic programs was derived in [21, 22] from their operational semantics. Here we choose to define \( \text{ERT} \) for quantum programs directly. Indeed, in the quantum case, \( \text{ERT} \) can also be derived from the operational semantics (see Section 3.2 of [39]), but the derivation is quite tedious. We believe that a direct definition can make our main idea clearer, and leave the derivation in the extended version [24] for interested readers.

We now present several useful properties of the function \( \text{ERT} \). The following lemma gives a way for computing the runtime of the iterations of a loop.

\[
\text{ERT}[\text{for} \ a \ \text{do} \ S][\rho] = \text{tr} \rho + \sum_{m \geq 0} \text{ERT}[\text{while}^{[m]}[M,S]][\rho]
\]
Lemma 2. For any input state \( \rho \), we have:

\[
\text{ERT[while}^{[n]} M, S \text{]}(\rho) = \sum_{k=0}^{n-1} \text{tr}\left( ([S] \circ \mathcal{E}_1)^k(\rho) \right) + \sum_{k=0}^{n-1} \text{ERT}[S](\mathcal{E}_1 \circ ([S] \circ \mathcal{E}_1)^k(\rho)).
\]

where \( \circ \) stands for the composition of super-operators.

Essentially, the \( k \)-th term of the first part in the right-hand side of the above equation is the time that the \( k \)-th measurement \( M \) takes in the process of iterations, and the \( k \)-th term of the second part is the time that the \( k \)-th application of \( S \) takes.

We define the set of inputs from which the expected runtime of program \( S \) is finite:

\[ T_S = \{ \rho \in \mathcal{D}(\mathcal{H}_S) : \text{ERT}[S](\rho) < \infty \}. \]

Then using Lemma 2, we derive:

\[
\text{ERT}[Q_{W}(\rho)] \geq \sum_{k=1}^{\infty} \text{tr}\left( ([S] \circ \mathcal{E}_1)^k(\rho) \right)
= \text{tr} \rho + \sum_{k=1}^{\infty} \text{tr}(\mathcal{E}_0 \circ ([S] \circ \mathcal{E}_1)^k(\rho))
\geq 1 + \sum_{k=1}^{\infty} \frac{1}{\pi}(1 - \frac{2}{\pi}) = \infty.
\]

Example 3.3. Consider quantum program \( Q \equiv Q_1; Q_2 \) defined in Eq. (6) with input \( q := |R \rangle \) and \( p := |1 \rangle \). We show that similar to its probabilistic counterpart \( C \equiv C_1; C_2 \), it has an infinite expected runtime. Let \( \rho_0 = |R\rangle\langle R| \otimes |1 \rangle \rho(1) \). It can be shown in a way similar to Example 3.1 that \( \text{ERT}[Q_1](\rho_0) = 7 \).

Moreover, it can be proved by induction that

\[
||Q_1||(\rho_0) = \sum_{k=1}^{\infty} \frac{1}{2^k} \langle L \rangle_q(\langle L | \otimes |2^k \rangle_p(2^k)).
\]

Then we have \( \text{ERT}[Q_2](\langle L \rangle_q(\langle L | \otimes |t \rangle_p(t)) = 2t + 1 \) for \( t > 0 \). Finally, we obtain:

\[
\text{ERT}[Q_1; Q_2](\rho_0) = 7 + \sum_{k=1}^{\infty} \frac{1}{2^k} \langle 2^{k+1} + 1 \rangle = \infty.
\]

4 Expected Runtime as an Observable

In the previous section, the expected runtime function \( \text{ERT}[S] \) of a quantum program \( S \) is simply seen as a real-valued function from input states. This is consistent with the idea of expected runtimes as generalised weakest preconditions in [21, 22] for probabilistic programs. However, quantum weakest preconditions are defined in [15] as physical observables. The aim of this section is to present a representation of expected run time \( \text{ERT}[S] \) as a physical observable, through which the close connection between \( \text{ERT}[S] \) and quantum weakest preconditions becomes clear. Such a representation brings an additional benefit. Obviously, it is inefficient to compute \( \text{ERT}[S](\rho) \) by Definition 3.1 for each input \( \rho \), especially when \( S \) contains loops. The representation of \( \text{ERT}[S] \), however, enables us to find a uniform way of computing expected runtime \( \text{ERT}[S](\rho) \) for all inputs \( \rho \), which will be presented in the next section.

4.1 Representation Theorem

The notion of observable is a cornerstone of quantum mechanics. Recall from [28] (or any standard textbook of quantum mechanics) that an observable of a quantum system with state Hilbert space \( \mathcal{H} \) is mathematically described by a Hermitian operator \( A \) on \( \mathcal{H} \). It determines a quantum measurement with the eigenvalues of \( A \) being the possible outcomes. If the system’s current state is \( \rho \) and we perform the measurement on it, then nondeterminism may happen.
here: different outcomes may occur with certain probabilities. But we have a statistical law; that is, one can assert that the expectation (average value) of the outcomes is $\text{tr}(A_S\rho)$.

Now let us consider how ERT(S) can be characterised in terms of physical observable. First of all, we notice that for each quantum program $S$, the linearity of ERT(S) shown in Lemma 3 immediately implies that there exists an observable $A_S$ such that

$$\text{ERT}(S)(\rho) = \text{tr}(A_S\rho)$$

(11)

for all $\rho \in T_S$ (the set of input states with finite expected runtimes; see Eq. (10)). So, our actual problem is to find an explicit and structural representation of this observable $A_S$.

To present the representation of $A_S$, we need the following lemma proved in [41]. Recall from [38] that $\text{tr}(\{|S\rangle\langle\psi|\}(\rho))$ is the probability that quantum program $S$ terminates with input $\rho$. Then for a density operator $\rho$, we say that $S$ almost surely terminates up to $\rho$ if $\text{tr}(\{|S\rangle\langle\psi|\}(\rho)) = 1$. By linearity of trace and $\{|S\rangle\}$, this condition should be rewritten as $\text{tr}(\{|S\rangle\langle\psi|\}(\rho)) = \text{tr}(\rho)$ if $\rho$ is a partial density operator.

**Lemma 4 (Theorem 3.1 in [41]).** For any quantum while-program with state Hilbert space $\mathcal{H}$, the set of the (unnormalised) pure states from which $S$ almost surely terminates:

$$V_S = \{|\psi\rangle \in \mathcal{H} : \text{tr}(\{|S\rangle\langle\psi|\}) = \text{tr}(\langle\psi|\psi\rangle)\}$$

(12)

is a subspace of $\mathcal{H}$, which can be computed using the Kraus operator-sum representation of $\{|S\rangle\}$.

We will also need the notion of the dual of a super-operator. For any (completely positive) super-operator $\mathcal{E}$ on Hilbert space $\mathcal{H}$ such that

$$\mathcal{E}(\rho) = \sum_j E_j \rho E_j^\dagger,$$

its dual $\mathcal{E}^\dagger$ is an operator on Hermitian operators defined by

$$\mathcal{E}^\dagger(A) = \sum_j E_j^\dagger A E_j.$$

The duality between operators $\mathcal{E}$ and $\mathcal{E}^\dagger$ is characterised by the equation $\text{tr}(\mathcal{A}\mathcal{E}(\rho)) = \text{tr}(\mathcal{E}^\dagger(A)\rho)$. Now we are ready to define the promised representation of observable $A_S$.

**Definition 4.1.** Let $\mathcal{A}$ be an observable. Then the expected runtime of quantum program $S$ followed by a continuation with expected runtime represented by $\mathcal{A}$ is inductively defined as follows:

1. $\text{ert}[\text{skip}] = \mathcal{A}$.
2. $\text{ert}[q := 0] = I + \sum_n |n\rangle_q\langle 0|A|0\rangle_q\langle n|$.
3. $\text{ert}[\widehat{\mathcal{U}}] = \mathcal{U}[\mathcal{U}]$.
4. $\text{ert}[S_1; S_2] = \text{ert}[S_1] \text{ert}[S_2]$.
5. $\text{ert}[\text{if}(\top m \cdot M[\overline{\mathcal{U}}] = m \rightarrow S_m)\mathcal{A}] = I + \sum_m \mathcal{E}^\dagger_m \mathcal{E}_m$ ($\mathcal{E}_m$ = $\text{ert}[S_m] = \mathcal{A}$).
6. $\text{ert}[\text{while} M[\overline{\mathcal{U}}] = 1 \text{ do } S \text{ od}] = \sum_{k=0}^\infty P_{\text{while}}(\mathcal{E}^\dagger_k \circ \{[S]\}^k(I)P_{\text{while}} + \mathcal{E}_0(\mathcal{A})) + \sum_{k=0}^\infty P_{\text{while}}(\mathcal{E}^\dagger_k \circ \{[S]\}^k \mathcal{E}_k(\text{ert}[S] \mathcal{E}_0(\mathcal{A})))P_{\text{while}}$

where $P_{\text{while}}$ is the projection onto the almost surely terminating subspace $V_{\text{while}}$ of loop while $\equiv$ while $M[\overline{\mathcal{U}}] = 1$ do $S$ od, as defined in Lemma 4.

We note that $\text{ert}[\text{while}][A]$ always converges, as guaranteed by the following lemma.

**Lemma 5.** Let while $\equiv$ while $M[\overline{\mathcal{U}}] = 1$ do $S$ od be a quantum loop and $X$ be a positive operator. Then

$$\sum_{k=0}^\infty P_{\text{while}}(\mathcal{E}^\dagger_k \circ \{[S]\}^k(X)P_{\text{while}}$$

converges.

Similar to classical and probabilistic weakest-precondition style calculus, Definition 4.1 is compositional for sequential programs. The main difference between $\text{ert}[S](\mathcal{A})$ and its probabilistic counterpart given in Table 1 of [22] is that the former is a Hermitian operator, and the latter is a real-valued function. In particular, the unit of time in their definitions is represented by the identity operator $I$ and the constant 1, respectively. On the other hand, from Section 4 of [15] and Section 4.2.2 of [39], it is clear that whenever the unit of time vanishes, $\text{ert}[S](\mathcal{A})$ degenerates to the weakest precondition $\text{wp.S.A}$ of $S$ given postcondition $A$.

By induction on the structure of $S$, it is easy to show that $\text{ert}[S](\mathcal{A})$ defined above is a positive (and thus Hermitian) operator. Therefore, it is indeed (the mathematical description of) a physical observable. In particular, the observable $\text{ert}[S](\mathcal{A})$ describes the expected runtime of program $S$ without any continuation. To simplify the presentation we abbreviate it to $\text{ert}[S]$ in the absence of ambiguity.

The following lemma clarifies the relationship between $\text{ert}[S](\mathcal{A})$ and $\text{ert}[S]$.

**Lemma 6.** For any observable $\mathcal{A}$ and initial state $\rho \in W_S$, we have:

$$\text{tr}(\text{ert}[S](\mathcal{A}) \cdot \rho) = \text{tr}(\text{ert}[S] \cdot \rho + \text{tr}(\mathcal{A} \cdot \{[S]\}(\rho)))$$

Now we can present the main result of this section. Note that all elements of the almost surely terminating subspace $V_S$ defined by Eq. (12) are pure states. We further define the set of partial density operators upon which program $S$ almost surely terminates:

$$W_S = \{\rho \in \mathcal{D}(\mathcal{H}) : \text{tr}(\{[S]\}(\rho)) = \text{tr}\rho\}.$$

Then our representation of $A_S$ can be stated as the following:

**Theorem 1 (Observable Representation of Runtime).** For any quantum program $S$ and all $\rho \in W_S$, we have:

$$\text{ERT}[S](\rho) = \text{tr}(\text{ert}[S] \cdot \rho).$$

(13)

The above theorem shows that observable $\text{ERT}[S]$ is an explicit representation of $A_S$ defined in Eq. (11) over $W_S$.

### 4.2 A Condition for Finite Expected Runtimes

The reader must have already noticed a gap between Eqs. (11) and (13): the former is valid over $T_S$ and the latter is valid over $W_S$. This gap can actually be filled in by the following:

**Theorem 2 (Equivalence of Almost Sure Termination and Finite Expected Runtime).** Let $S$ be an arbitrary quantum while-program with state Hilbert space $\mathcal{H}$. Then for any partial density operator $\rho \in \mathcal{D}(\mathcal{H})$:

$$\text{ERT}[S](\rho) < \infty \iff \text{tr}(\{[S]\}(\rho)) = \text{tr}\rho.$$
An immediate corollary of the above theorem is $T_S = W_S$. Moreover, it shows that a quantum program $S$ with input $\rho$ almost surely terminates if and only if it has a finite expected runtime.

The above theorem further implies that if $\rho \in W_S$, then $\text{ERT}[S](\rho) = \text{tr}(\text{ert}[S] \cdot \rho)$; otherwise $\text{ERT}[S](\rho) = \infty$. However, calculating the value of $\text{ert}[S]$ is challenging when $S$ has loops. This problem will be considered in Section 5.2.

Remember that in this paper, we only consider quantum programs in finite-dimensional Hilbert spaces. Then we can furthermore give a finite upper bound for the expected runtime for a quantum program $S$ for all almost surely terminating inputs $\rho \in W_S$. Recall that for an operator $A$ on a Hilbert space $\mathcal{H}$, its norm is defined as

$$\|A\| = \sup\{|A|\psi\| : |\psi\rangle \in \mathcal{H} \text{ and } ||\psi\rangle = 1\}.$$  

**Corollary 4.1 (Uniform Bound of Expected Runtime).** For any quantum program $S$, let $M$ be the norm of its expected runtime observable: $M = \|\text{ert}[S]\|$. Then for all $\rho \in W_S$, we have

$$\text{ERT}[S](\rho) \leq M.$$  

## 5 Computing Expected Runtime

Theorem 1 established in the last section gives a physical interpretation of expected runtime as an observable. As pointed out before, this observable representation provides us with an efficient method for computing expected runtimes. More precisely, we see from Eq. (13) that if the observable $\text{ert}[S]$ is known, we can easily compute the expected runtime $\text{ERT}[S](\rho)$ for any input $\rho$. Therefore, in this section, we develop a method for computing $\text{ert}[S]$ based on the matrix representation of denotational semantics of quantum programs, by extending and incorporating the approach of [40] for termination analysis into the setting of expected runtimes.

### 5.1 Matrix Representation of Super-Operators

Recall from Lemma 1 that the denotational semantics of a quantum program is a (completely positive) super-operator mapping partial density operators (i.e., matrices in the finite-dimensional case considered in this paper) to themselves. For the convenience of the reader, in this subsection we briefly review some necessary mathematical tools for computing super-operators from [39] (see Section 5.1.2 there).

Let us first show that an $d \times d$ matrix can be encoded as an $d^2$-dimensional vector using a maximally entangled state. Assume that $\{|j\rangle\}_j$ is an orthonormal basis of $\mathcal{H}$. Then we write: $|\Psi\rangle = \sum_j |jj\rangle$ for the (unnormalised) maximally entangled state in $\mathcal{H} \otimes \mathcal{H}$.

**Lemma 7.** For any $d \times d$ matrix $A = (A_{ij})$, we have:

$$(A \otimes I)|\Psi\rangle = (A_{00}, A_{01}, \ldots, A_{0(d-1)}, A_{10}, \ldots, A_{(d-1),0}, A_{(d-1),1}, \ldots, A_{(d-1),(d-1)}).$$  

Note that the left-hand side of Eq. (14) is a $d^2$-dimensional vector. As is well-known that in practice, an abstract super-operator is usually hard to compute. But we can often use its matrix representation in the computation.

### Definition 5.1

Suppose super-operator $E$ on a Hilbert space $\mathcal{H}$ has the Kraus operator-sum representation:

$$E(\rho) = \sum_m M_m \rho M_m^\dagger$$  

for all $\rho \in D(\mathcal{H})$. Then its matrix representation is defined as the following operator on $\mathcal{H} \otimes \mathcal{H}$:

$$M_E = \sum_m M_m \otimes M_m^\dagger$$

where $M_m^\dagger$ is the conjugate of $M_m$, and $\otimes$ stands for tensor product.

Note that when the dimension $d = \dim \mathcal{H} < \infty$, each $M_m$ in Eq. (15) is a $d \times d$ matrix, and thus $M_E$ is a $d^2 \times d^2$ matrix.

The next lemma gives a close connection between a super-operator $E$ and its matrix representation through the maximal entanglement.

**Lemma 8.** For any $d \times d$ matrix $A$, we have:

$$\langle E(A) \otimes I | \Psi \rangle = M_E(A \otimes I) |\Psi\rangle$$  

We observe that in the left-hand side of Eq. (16) super-operator $E$ is applied to operator $A$, but all operations on the right-hand side are matrix multiplications. As we will see below, a combination of the above two lemmas provides us with an effective way of manipulating super-operators in computing expected runtime of quantum programs.

### 5.2 Computing the Expected Runtimes of Loops

We see from Definition 4.1 that computing runtime observable $\text{ert}[S]$ is difficult only when $S$ contains while-loop. So, this subsection is devoted to develop a method for computing the following runtime observable of loop:

$$\text{ert}[\text{while } M[\overline{\gamma}] = 1 \text{ do } S \text{ od}]$$

using the matrix representation introduced in the previous subsection. For simplicity, we write while for the loop

$$\text{while } M[\overline{\gamma}] = 1 \text{ do } S \text{ od}.$$  

Assume that $\text{ert}[S]$ is given. First, we notice that computing $\text{ert}[\text{while}]$ directly using Definition 4.1 is very difficult because (possibly infinitely) many iterations of super-operators are involved there.

However, this difficulty can be circumvented with the matrix representations of these super-operators. More precisely, repeatedly using Lemma 8, we obtain:

$$(\text{ert}[\text{while}] \otimes I)|\Psi\rangle = \sum_{k=0}^{\infty} (E_P \circ (E_1^* \circ [S]^\dagger)^k)(I + E_1^*(\text{ert}[S]) \otimes I)|\Psi\rangle$$  

where as in Definition 4.1, $P$ is the projection onto the almost surely terminating subspace $V_{\text{while}}$ of loop while, and $E_P(\rho) = P \rho P^\dagger$, $E_1(\rho) = M_1 \rho M_1^\dagger$

for all $\rho$, $M_1 = P \otimes P^\dagger$ is the matrix representation of $E_P$, and $[S]^\dagger$ is the dual of super-operator $[[S]]$ (the denotational semantics of $S$).
Note that the super-operators in the infinite series of (18) are all transferred to their matrix representations in (19).

Next we compute the infinite series of matrices in (19). Let us introduce matrix:

$$R = M_\mathcal{E}_1M_{\mathcal{I}[S]}.$$ 

Then this infinite series can be simply written as:

$$\sum_{k=0}^\infty M_p R^k.$$ (20)

We see from Lemma 5 that (20) always converges. If the program while almost surely terminates on all initial states (i.e., $P = I$), then (20) can be computed directly by

$$\sum_{k=0}^\infty M_p R^k = \sum_{k=0}^\infty R^k = (I \otimes I - R)^{-1}.$$ (21)

In the case that $P \neq I$, the series (20) can be computed using the Jordan decomposition-based technique introduced in [40].

Suppose that the Jordan decomposition of $R$ is

$$R = A J(R) A^{-1}$$

where $J(R)$ is the Jordan normal form of $R$ such that

$$J(R) = \sum_{i=1}^l J_k(\lambda_i) = \text{diag}(J_{k_1}(\lambda_1), J_{k_2}(\lambda_2), \ldots, J_{k_l}(\lambda_l))$$

where $J_k(\lambda_i)$ is a $k_i \times k_i$-Jordan block of eigenvalue $\lambda_i$. Since all super-operators considered in this paper do not increase the trace: $tr(E(\rho)) \leq tr(\rho)$ for partial density operators $\rho$, we have:

**Lemma 9 (cf. Lemma 4.1 in [40]).**

1. The eigenvalues satisfy: $|\lambda_i| \leq 1$ for $1 \leq i \leq l$.
2. If $|\lambda_i| = 1$, then the dimension of the $i$th Jordan block is $k_i = 1$.

It is known from matrix analysis [19] that whenever some eigenvalue $\lambda_i$ has module 1, $\sum_{k=0}^\infty R^k$ will be diverging. Fortunately, we can remove these eigenvalues by changing $R$ to

$$N = AJ(N)A^{-1}$$

where the Jordan normal form $J(N)$ of $R$ is replaced by

$$J(N) = \text{diag}(J'_k(\lambda_1), J'_k(\lambda_2), \ldots, J'_k(\lambda_l))$$

where

$$J'_k(\lambda_i) = \begin{cases} 0 & \text{if } |\lambda_i| = 1, \\ J_k(\lambda_i) & \text{otherwise} \end{cases}$$

that is, $J'_k(\lambda_i)$ is the same as the Jordan block $J_k(\lambda_i)$ of $J(R)$ when the module of its eigenvalue is less than 1, but whenever eigenvalue $\lambda_i$ has module 1, then the corresponding 1-dimensional Jordan block is simply replaced by 0. The following lemma guarantees that such a modification is feasible:

**Lemma 10 (cf. Lemma 4.2 in [40]).** $M_p R^k = M_p N^k$ for all integers $k \geq 0$.

As a result of the above lemma, the infinite series of matrices in equation (19) can be computed as follows:

$$\sum_{k=0}^\infty M_p R^k = M_p \sum_{k=0}^\infty N^k = M_p (I \otimes I - N)^{-1}.$$ (22)

Plugging (22) into (19), we obtain:

$$\text{ert[while] } \otimes I |\Psi\rangle$$

$$= \sum_{k=0}^\infty M_p R^k \cdot \{ (I + E_1^\ast(\text{ert}[S])) \otimes I \} |\Psi\rangle$$

$$= M_p (I \otimes I - N)^{-1} \cdot \{ (I + E_1^\ast(\text{ert}[S])) \otimes I \} |\Psi\rangle.$$ (23)

Since $\text{ert}[S]$ is assumed to be given, $(\text{ert[while]} \otimes I) |\Psi\rangle$ is computed now. Furthermore, using Lemma 7, $\text{ert[while]}$ can be retrieved from $(\text{ert[while]} \otimes I) |\Psi\rangle$.

To conclude this section, we remark that for an arbitrary quantum program $S$, its expected runtime $\text{ERT}[S]$ can be inductively computed by the above procedure combined with Definition 4.1. Furthermore, by Theorems 1 and 2, either $\text{ERT}[S](\rho) = \infty$, or it can be computed as $\text{ERT}[S](\rho) = tr(\text{ert}[S] \cdot \rho)$.

### 6 CASE STUDIES

In this section, we present two more sophisticated examples to show the power of our method for computing the expected runtimes of quantum programs developed in the last section.

#### 6.1 Quantum Bernoulli Factory

Classical Bernoulli Factory (CBF) is an algorithm for generating random numbers. More precisely, it simulates a new coin that has probability $f(p)$ of heads given a coin with unknown probability $p$ of heads, where $f : [0, 1] \rightarrow [0, 1]$ is a function.

Quantum Bernoulli Factory (QBF) was proposed in [14] as a quantum counterpart of CBF. Comparing to CBF, QBF can utilize a quantum coin like $|p\rangle = \sqrt{|0\rangle + \sqrt{1-p}|1\rangle}$. It was proved in [14] that QBF can simulate a strictly larger class of function $f$ than CBF.

**Quantum Program QBF:** An example that QBF can simulate but CBF cannot is:

$$f(p) = 1 - |2p - 1| = \begin{cases} 2p & p \in [0, 1/2]; \\ 2(1-p) & p \in (1/2, 1]. \end{cases}$$

The key of simulating $f$ is to simulate $f'(p) = (2p - 1)^2$. To this end, we construct a quantum coin

$$|f'(p)\rangle = (2p - 1)|0\rangle + 2\sqrt{p(1-p)}|1\rangle$$

using the following program

$$\text{QBF} \equiv q_1 := |1\rangle; q_2 := |1\rangle; \text{ while } M[q_2] = 1 \text{ do } S \text{ od}$$ (24)

where $M = \{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ is the measurement on a qubit in the computational basis, and the loop body is

$$S \equiv q_1 := |p\rangle; q_2 := |p\rangle; q_1, q_2 := U[q_1, q_2]$$ (25)

with unitary transformation:

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & -\frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & -\frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{pmatrix}.$$
We were able to show in Example 3.2 by Definition 3.1 that the quantum walk on an 2-loop in QBF. Let

\[ W \equiv \text{while } M[q_2] = 1 \text{ do } S \text{ od,} \]

da direct application of formula (23) yields:

\[
(ert[W] \otimes I)\Psi = (I \otimes I - M_{c_{1}}M_{[S]})^{-1}(I \otimes I)\Psi \\
+ (I \otimes I - M_{c_{1}}M_{[S]})^{-1}(E_{1}^{-1}(ert[S] \otimes I)I)\Psi
\]

The left-hand side of the above equation is the vector representation of \( ert[W] \) (see Lemma 7). The first term in the right-hand side is the expected time that measurement \( M \) takes, and the second term is the expected runtime of the loop body takes.

It is easy to see that \( ert[S] = 5 \cdot I \) by the definition of \( ert \). Therefore, the runtime observable is:

\[
ert[W] = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 3
\end{pmatrix}
\]

Hence, by Definition 4.1 we obtain the runtime observable of QBF: \( ert[QBF] = 17 \cdot I \), where \( I \) is the identity operator. Thus, the expected runtime of QBF is \( ERT(QBF)(\rho) = 17 \) for all density operators. It is interesting to see that the expected runtime is independent of the probabilistic parameter \( p \).

### 6.2 Quantum Random Walk

We were able to show in Example 3.2 by Definition 3.1 that the expected runtime of a simple quantum random walk, namely Hadamard walk, initialized in direction \( L \) and position 1, is \( \infty \). In this subsection, we consider a more complicated quantum random walk, a quantum walk on an \((n+1)\)-circle with absorbing boundaries at positions 0 and \( n \), of which the expected runtime is hard to compute using Definition 3.1 directly. Instead, we will compute it using the method introduced in the previous section.

**Quantum Program QW:** The quantum coin is the same as before, with \( H_{q} \) spanned by the orthonormal basis \( \{|0\}, |1\} \) as its state Hilbert space. But the position space is an \( n \)-dimensional Hilbert space \( H_{p} \) with orthonormal basis \( \{|0\}, |1\}, \ldots, |n\} \), where basis state \( |i\rangle \) is used to denote position \( i \) on the circle. The state space of quantum walk is \( H \equiv H_{q} \otimes H_{p} \). Each step of the quantum walk consists of:

1. Measure the position of the system to see whether it is the absorbing boundary 0 or \( n \). If it is the case, the walk terminates; otherwise, it continues. Mathematically, the measurement is described as \( M = \{M_{0}, M_{1}\} \), where

\[
M_{0} = |0\rangle \langle 0| + |n\rangle \langle n|, \quad M_{1} = |p \rangle \langle p|.
\]

2. Toss the coin by applying an operator \( T \) on \( H_{q} \):

\[
T = \begin{pmatrix}
a & b^{*} e^{i\phi} \\
b & -a^{*} e^{i\phi}
\end{pmatrix}
\]

where \( a, b \) are complex numbers satisfying the normalisation condition: \( |a|^{2} + |b|^{2} = 1 \). Note that the coin tossing operator here is a general \( 2 \times 2 \) unitary operator rather than the Hadamard gate \( H \).

3. Shift the position to the left or right according to the state of coin. The shift operator is given as

\[
S = \sum_{i=0}^{n} |L \rangle \langle i| + |R \rangle \langle i| + \sum_{i=0}^{n} |R \rangle \langle L| + |i \rangle \langle i|
\]

where \( \otimes \) and \( + \) are addition and subtraction modulo \( n + 1 \). Note that addition and subtraction modulo \( n + 1 \) are used here because the walk is on an \((n + 1)\)-circle.

The above process can be formally described as the following quantum loop:

\[
QW_{n} \equiv \text{while } M[p] = 1 \text{ do } q := T[q]; q, p := S[q, p] \text{ od}
\]

**Expected Runtime of QW:** The expected runtime of \( QW_{n} \) has been an open problem since [4], and it was proved in [40] to be \( n \) for a special initial state with \( n < 30 \). Here, we compute \( ERT(QW_{n}) \) for the general case using the method developed in the previous section. To this end, let us write:

\[
QW'_{n} \equiv q := T[q]; q, p := S[q, p]
\]

for its loop body. It was shown in Theorem 4.5 of [23] that \( QW_{n} \) almost surely terminates on all computational basis states. Based on this, we can prove that \( QW'_{n} \) almost surely terminates on all initial states by Lemma 4, and thus has a finite expected runtime by Theorem 2. Then by formulas (25) and (21), the runtime observable of \( QW'_{n} \) can be computed as follows:

\[
(ert[QW'_{n}] \otimes I)\Psi = (I \otimes I - E \otimes E)^{-1}(I \otimes I)\Psi \\
+ (I \otimes I - E \otimes E)^{-1}(E_{1}^{-1}(ert[QW'_{n}] \otimes I)I)\Psi
\]

where

\[
E = M_{1}^{T}(T \otimes I)^{T} S.
\]
We are more interested in the first term in the right-hand side of the above equation because it is actually the expected steps that the quantum random walk goes plus one. Let

\[ Q_n = \sum_{k=0}^{\infty} (E^T \circ ||QW^k||^2)^k (I). \]

Then we have:

\[ (Q_n \otimes I) |\Psi\rangle = (I \otimes I - E \otimes E^\dagger)^{-1} |\Psi\rangle. \]

Consequently, \((L, k|Q_n|L, k)\) and \((R, k|Q_n|R, k)\) are exactly the expected steps that the quantum random walk takes when it begins with states \([L, k]\) and \([R, k]\), respectively.

In practice, it is a bit difficult to calculate \(Q_n\) if \(n\) is treated as an abstract parameter. However, we can easily calculate \(Q_n\) when \(n\) is a given integer. Moreover, we can guess a pattern \(X\) of \(Q_n\) for arbitrary \(n\) from these results of some given values of \(n\). Then, it is sufficient to show that the pattern that we guessed is exactly \(Q_n\). Suppose that we have a matrix \(X\) such that

\[ I + EXE^\dagger = X. \]

Then we obtain:

\[ (X \otimes I) |\Psi\rangle = ((I + EXE^\dagger) \otimes I) |\Psi\rangle = |\Psi\rangle + (E \otimes E^\dagger) (X \otimes I) |\Psi\rangle \]

by Lemma 8. The previous equation is equivalent to

\[ (I \otimes I - E \otimes E^\dagger) (X \otimes I) |\Psi\rangle = |\Psi\rangle \]

Since this quantum random walk is almost surely terminating, \((I \otimes I - E \otimes E^\dagger)\) is invertible. Thus, we have:

\[ (Q_n \otimes I) |\Psi\rangle = (X \otimes I) |\Psi\rangle. \]

Note that \((Q_n \otimes I) |\Psi\rangle\) is an encoding of \(Q_n\) according to Lemma 7. Then we can conclude that \(Q_n = X\) by Lemma 7. Now we compute \(Q_n\) in the following four steps:

**Step 1: Reduce to real coin tossing operators.** The entries of coin tossing operator \(T\) are allowed to be complex numbers. But we can show that it is sufficient to deal with the case where all entries of \(T\) are reals. Since \(T\) is unitary, by the Z-Y decomposition (see Theorem 4.1 in [28]), we can find reals \(x, y, \alpha, \beta, \delta\) so that

\[ T = e^{i\alpha} \begin{pmatrix} e^{-i(\beta+\delta)x} & e^{-i(\beta-\delta)y} \\ e^{i(\beta-\delta)y} & e^{i(\beta+\delta)x} \end{pmatrix} x^2 + y^2 = 1. \]

Note that \(EXE^\dagger\) is irrelevant to \(\alpha\), we can assume \(\alpha = 0\) here.

Consider another quantum walk that uses the coin-tossing operator:

\[ T' = \begin{pmatrix} x & y \\ y & -x \end{pmatrix} \]

Note that all entries of matrix \(T'\) are reals. Let

\[ E' = M_1^T (T' \otimes I) M_1^\dagger S^\dagger, \]

\[ (Q_n' \otimes I) |\Psi\rangle = (I \otimes I - E' \otimes E'^\dagger)^{-1} |\Psi\rangle. \]

Then we have:

**Lemma 11.** \(Q_n\) and \(Q'_n\) are related by a unitary operator \(P\):

\[ Q_n = P^\dagger Q'_n P \]

where:

\[ P = \begin{pmatrix} P_L & 0 \\ 0 & P_R \end{pmatrix}. \]

\[ P_L = \sum_{k=0}^{n} e^{-i[(\beta+\delta)k+2\delta]} |k\rangle \langle k|, \]

\[ P_R = \sum_{k=0}^{n} e^{-i(\beta+\delta)k} |k\rangle \langle k|. \]

With the above lemma, we only need to consider the case where \(\phi = 0\) and \(a, b\) are both reals.

**Step 2: Find the pattern of \(Q_n\):** To this end, we first compute \(Q_n\) with the Hadamard coin-tossing operator \(T = H\) for some fixed \(n\)’s. Our results show that \(Q_n\) has the form:

\[ Q_n = \begin{pmatrix} A_n & B_n^* \\ B_n & C_n \end{pmatrix} \]

(27)

where \(A_n, B_n\) and \(C_n\) are all \((n+1) \times (n+1)\) matrices. Moreover, we see that the non-zero entries \((A_n)_{i,j}\) and \((C_n)_{i,j}\) in \(A_n\) and \(C_n\) are all quadratic polynomials of \(i\) and \(j\), and the non-zero entries \((B_n)_{i,j}\) in \(B_n\) are linear in \(i\) and \(j\).

**Step 3: Solution of \(Q_n\):** Finally, we can present a solution of \(Q_n\) for a general coin tossing operator \(T\) with real entries. We use pattern (27), so what we need to compute are matrices \(A_n, B_n\), and \(C_n\). Let

\[ f_n(j, k) = (-1)^{\frac{j-k}{2}} \cdot (a^2 - (j \mod n) \cdot (n - 1) - k), \]

\[ h_n(j, k) = (-1)^{\frac{j-k}{2}} \cdot b \cdot (j + k - n). \]

The solutions of \(A_n, B_n\) and \(C_n\) can be given as follows:

\[ (A_n)_{j,k} = \begin{cases} f_n(j, k) + 1 & \text{if } j = k, \\ f_n(j, k) + (j \mod n) & \text{if } 0 < j < k < n \text{ and } 2 | (j - k), \\ f_n(j, k) & \text{if } 0 < j < k < n \text{ and } 2 \not| (j - k), \\ 0 & \text{otherwise}, \end{cases} \]

\[ (B_n)_{j,k} = \begin{cases} h_n(j, k) & \text{if } 0 < j < k < n \text{ and } 2 | (j - k), \\ 0 & \text{otherwise}, \end{cases} \]

\[ (C_n)_{j,k} = (A_n)_{n-j,n-k}. \]

**Step 4: Verification of \(Q_n\):** The correctness of the above solutions can be verified by checking the following equation:

\[ \begin{pmatrix} A_n & B_n^* \\ B_n & C_n \end{pmatrix} = I + E \begin{pmatrix} A_n & B_n^* \\ B_n & C_n \end{pmatrix} E^\dagger \]

\[ = I + \begin{pmatrix} M_1 & 0 \\ 0 & M_1 \end{pmatrix} \begin{pmatrix} aS_L^2 & bS_L \\ bS_L & -aS_L \end{pmatrix} \begin{pmatrix} A_n & B_n^* \\ B_n & C_n \end{pmatrix} \]

\[ = \begin{pmatrix} M_1 & 0 \\ 0 & M_1 \end{pmatrix}, \]

where \(S_L = \sum_{i=0}^{n} (i \otimes 1) (i). \) This equation can be divided into four equations for the sub-matrices. Each of them can be checked directly.
by matrix calculation. As an example, we check the equation for the top left submatrix, which is

\[ A_n = I_n + M_1 (a^2 S_L^T A_n S_L + b^2 S_L C_n S_L^T)
+ ab S_L B_n S_L + ab S_L^T B_n^T S_L) M_1 \]

According to the construction of \( B_n \), there are two cases which depend on whether the element of \( A_n \) is on the diagonal. These two cases can be reduced to the equations:

\begin{align*}
(A_n)_{j, j} &= 1 + a^2 \cdot (A_n)_{j-1, j-1} + b^2 \cdot (A_n)_{n-(j+1), n-(j+1)}, \\
(A_n)_{j, k} &= a^2 \cdot (A_n)_{j-1, k-1} + b^2 \cdot (C_n)_{j+1, k+1} - ab \cdot (B_n)_{j+1, k-1}
\end{align*}

(28)

for \( 0 < j < k < n \). Both of the above equations can be checked by simple calculation. As a result, we have:

**Proposition 6.1.** The expected steps of \( QW_n \) starting from state \( |L, k \rangle \) and \( |R, k \rangle \) are:

\begin{align*}
(L, k)QW_n|L, k \rangle &= f_n(k, k) + (k \mod n) + 1, \\
(R, k)QW_n|R, k \rangle &= f_n(n - k, n - k) + ((n - k) \mod n) + 1
\end{align*}

respectively. More generally, if \( \Psi \) starts from

\[ \Psi = \sum_{k=0}^{n} (\alpha_k |L, k \rangle + \beta_k |R, k \rangle) \]

the expected step of \( QW_n \) is:

\[ \langle \Psi | QW_n | \Psi \rangle = \sum_{j=0}^{n} \left( f_n(j, j) + (j \mod n) + 1 \right) \]

\[ \times \left( a_j^* \alpha_j + \beta_{n-j}^* \beta_{n-j} \right) + \sum_{j=0}^{n-1} \left( h_n(j, j)(a_j^* \beta_j + \beta_j^* \alpha_j) \right) \]

\[ + \sum_{0 < j < k < n, (k - j)} \left( h_n(j, k)(\beta_j^* \alpha_k + \alpha_j^* \beta_k) \right) \]

\[ + f_n(j, k) \left( a_j^* \alpha_k + \alpha_j^* \alpha_k + b_j^* \beta_n - \beta_n^* \alpha_j \right) \]

It deserves to mention that the first term in Eq. (28) coincides with the expected runtime of classical random walk on an \( n \)-circle which moves to \( k - 1 \) with probability \( a^2 \) and moves to \( n - k + 1 \) with probability \( b^2 \) from position \( 0 < k < n \). For example, the expected steps taken by the quantum random walk on a 6-circle starting from state \( |L \rangle |k \rangle \) are equal to the expected runtime of the classical random walk in Figure 1 starting from \( k \) and terminating at \( t \).

7 CONCLUSION

In this paper, we defined the expected runtimes of quantum programs as a generalisation of quantum weakest precondition [15]. A representation of the expected runtimes as a quantum observable was presented. This representation gives a physical interpretation of the notion of expected runtime. Based on it, we develop an effective method for computing the expected runtimes of quantum programs in finite-dimensional Hilbert spaces using the mathematical tool of the matrix representation of super-operators. We demonstrated the power of our computational method through several case studies, including the expected runtime of quantum Bernoulli factory — a quantum algorithm for generating random numbers; in particular, our method is able to compute the expected runtime of quantum walk on an \( n \)-circle, for arbitrary \( n \), an arbitrary quantum coin and an arbitrary initial state, and thus solve an open problem.

The basic idea of this paper came from recent work on the corresponding problem for probabilistic programs [21, 22, 26], but the computational method presented in this paper is quite different from there. Except for a weakest precondition calculus, a set of proof rules for reasoning about the expected runtime of probabilistic programs was also presented in [21, 22]. It is also possible to develop some similar proof rules for quantum programs by extending quantum Hoare logic [38].

Our approach to computing the expected runtime of quantum programs is limited to the case of finite-dimensional Hilbert spaces and thus cannot deal with the quantum integer type. Nevertheless, most of the existing quantum algorithms have been designed in the finite-dimensional case, and our results can be applied to them. On the other hand, the infinite-dimensional case is certainly an interesting (and challenging, we believe) topic for future research.

We saw in Subsection 6.2 that the computation of the expected runtime of a quantum program can be much more involved than that of a probabilistic program. So, one of the most important topics for future research is efficient symbolic automation of our computational method; more specifically, for example, how to combine the quantum weakest precondition-style reasoning developed in this paper with the automatic amortised resource analysis (AARA) [8, 9, 17, 18, 26] for computing the expected runtimes of more complicated quantum programs.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (Grant No: 2018YFA0306701) and the National Natural Science Foundation of China (Grant No: 61832015).
REFERENCES

[1] Dorit Aharonov, Andris Ambainis, Julia Kempe, and Umesh Vazirani. 2001. Quantum walks on graphs. In Proceedings of the thirty-third annual ACM symposium on Theory of computing. ACM, Association for Computing Machinery, 50–59. https://doi.org/10.1145/380752.380758

[2] Andris Ambainis. 2007. Quantum Walk Algorithm for Element Distinctness. SIAM J. Comput. 37, 1 (2007), 210–239. https://doi.org/10.1137/S0097539704447311

[3] Andris Ambainis. 2008. Quantum random walks–new methods for designing quantum algorithms. In International Conference on Current Trends in Theory and Practice of Computer Science. Springer, Berlin Heidelberg, 1–4. https://doi.org/10.1007/978-3-540-75766-9_1

[4] Andris Ambainis, Eric Bach, Ashwin Nayak, Ashvin Vishwanath, and John Watrous. 2001. One-dimensional Quantum Walks. In Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing (Hersonissos, Greece) (STOC ’01). ACM, 37–46. https://doi.org/10.1145/380752.380757

[5] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando Brandao, David A. Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. Nature 574, 7779 (2019), 505–510. https://doi.org/10.1038/s41586-019-1666-5

[6] Kevin Bats, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2018. How long, O Bayesian network, will I sample thee? In European Symposium on Programming. Springer, Springer International Publishing, 186–213. https://doi.org/10.1007/978-3-95938-006-1_9

[7] Tomasz Bezold, Stefan Kiefer, Antonin Kuera, and Ivana Hutatova Vaˇtová. 2015. Runtime analysis of probabilistic programs with unbounded recursion. J. Comput. System Sci. 81, 1 (2015), 288–310. https://doi.org/10.1016/j.jcss.2014.06.005

[8] Quentin Carbonneaux, Jan Hoffmann, Thomas Reps, and Zhong Shao. 2017. Automated resource analysis with Coq proof objects. In International Conference on Computer Aided Verification. Springer, Springer International Publishing, 64–85. https://doi.org/10.1007/978-3-319-63390-9_4

[9] Orieta Celiku and Annabelle McIver. 2005. Compositional specification and analysis of cost-based properties in probabilistic programs. In International Symposium on Formal Methods. Springer, Springer Berlin Heidelberg, 107–122. https://doi.org/10.1007/11526841_9

[10] Pavol Cerny, Thomas A Henzinger, Laura Kovacs, Arjun Radhakrishna, and Jakob Zwirchmayr. 2015. Segment abstraction for worst-case execution time analysis. In European Symposium on Programming Languages and Systems. Springer, Springer Berlin Heidelberg, 105–131. https://doi.org/10.1007/978-3-662-46669-8_5

[11] Aleksandr Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic program analysis with martingales. In International Conference on Computer Aided Verification. Springer, Springer Berlin Heidelberg, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

[12] Krishnendu Chatterjee, Hongfei Fu, and Ankit Murhekar. 2017. Automated recurrence analysis for almost-linear expected-runtime bounds. In International Conference on Computer Aided Verification. Springer, Springer International Publishing, 118–139. https://doi.org/10.1007/978-3-662-53506-7_9

[13] Andrew Chros. 2018. The IBM Q experience and Qiskit open-source quantum computing software. In APS March Meeting Abstracts. Volume 2018. 158–A08. https://doi.org/10.1103/PhysRevA.97.052303

[14] Howard Dale, David Jennings, and Terry Rudolph. 2015. Provable quantum advantage in randomness processing. Nature communications 6 (2015), 2803. https://doi.org/10.1038/ncomms2803

[15] Elie D’Hondt and Prakash Panangaden. 2006. Quantum weakest preconditions. In Quantum Weakest Preconditions for Reasoning about Expected Runtimes of Quantum Programs LICS ’22, August 2–5, 2022, Haifa, Israel. 85. https://doi.org/10.1109/329679/3192394

[16] Jan Hoffmann, Quentin Carbonneaux, and Jan Hoffmann. 2018. Absorption probabilities of quantum walks. Quantum Information Processing 17, 10 (2018), 1–34. https://doi.org/10.1007/s11128-017-1817-4

[17] Junyi Liu, Li Zhou, Gilles Barthe, and Mingheng Ying. 2022. Quantum Weakest Preconditions for Reasoning about Expected Runtimes of Quantum Programs (Extended Version). arXiv preprint arXiv:1911.12557 (2022).

[18] Annabelle McIver, Carroll Morgan, and Charles Carroll Morgan. 2005. Abstraction, refinement and proof for probabilistic systems. Springer Science & Business Media.

[19] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded expectations: resource analysis for probabilistic programs. In ACM SIGPLAN Notices, Vol. 53. ACM, Association for Computing Machinery, 496–512. https://doi.org/10.1145/329679/3192394

[20] Van Chan Ngo, Mario Dehesa-Aznara, Matthew Fredrikson, and Jan Hoffmann. 2017. Verifying and Synthesizing Constant-Resource Implementations with Types. In 2017 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, 710–728. https://doi.org/10.1109/SP.2017.53

[21] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

[22] Hanne R. Nielsen. 1987. A hoare-like proof system for analysing the computation time of programs. Science of Computer Programming 9, 2 (1987), 107–136. https://doi.org/10.1016/0167-6423(87)90029-3

[23] Federico Olmedo and Alejandro Diaz-Caro. 2019. Runtime Analysis of Quantum Programs: A Formal Approach. arXiv preprint arXiv:1911.11247 (2019).

[24] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about recursive probabilistic programs. In 2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). IEEE, Association for Computing Machinery, 1–10. https://doi.org/10.1109/LICS.2016.28

[25] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. Quantum 2 (2018), 79. https://doi.org/10.22331/q-2018-08-06-79

[26] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Lauter. 2017. Quantum resource estimates for computing elliptic curve discrete logarithms. In International Conference on the Theory and Application of Cryptology and Information Security. Springer, Springer International Publishing, 241–270. https://doi.org/10.1007/978-3-540-75267-5_9

[27] J. Sanders and P. Zuliani. 2008. Quantum Programming. In Proceedings of the 5th International Conference on Mathematics of Program Construction (MPC 2000) (LNCS, Vol. 1837). Springer Berlin Heidelberg, 80–99. https://doi.org/10.1007/10722030_6

[28] Peter Selinger. 2004. Towards a quantum programming language. Mathematical Structures in Computer Science 14, 4 (2004), 527–586. https://doi.org/10.1017/S0960129503004025

[29] Neil Shenvi, Joseph K. Liu, and K Birgitta Whaley. 2003. Quantum random-walk search algorithm. Physical Review A 67, 5 (2003), 052307. https://doi.org/10.1103/PhysRevA.67.052307

[30] Pedro Vazconcelos, Steffen Jost, Mario Florido, and Kevin Hammond. 2015. Type-based allocation analysis for co-reursion in lazy functional languages. In European Symposium on Programming Languages and Systems. Springer, Springer Berlin Heidelberg, 787–811. https://doi.org/10.1007/978-3-662-46669-8_32

[31] Mingheng Ying. 2012. Floyd–Hoare Logic for Quantum Programs. ACM Trans. Program. Lang. Syst. 33, 6, Article 19 (2012). https://doi.org/10.1145/2049706.2049708

[32] Mingheng Ying. 2016. Foundations of Quantum Programming. Morgan Kaufmann.

[33] Mingheng Ying, Nengkun Yu, Yuan Feng, and Runyao Duan. 2013. Verification of quantum programs. Science of Computer Programming 78, 9 (2013), 1679–1700. https://doi.org/10.1016/j.scico.2013.03.016

[34] Li Zhou, Nengkun Yu, and Mingheng Ying. 2019. An Applied Quantum Hoare Logic. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019). Association for Computing Machinery, 1149–1162. https://doi.org/10.1145/3314221.3314584