Quantum Expectation Transformers for Cost Analysis

Martin Avanzini
Inria Sophia Antipolis - Méditerranée
France
martin.avanzini@inria.fr

Georg Moser
Department of Computer Science
Universität Innsbruck
Innsbruck, Austria
georg.moser@uibk.ac.at

Romain Péchoux
Université de Lorraine, CNRS, Inria,
LORIA, F-54000 Nancy
France
romain.pechoux@inria.fr

Simon Perdrix
Université de Lorraine, CNRS, Inria,
LORIA, F-54000 Nancy
France
simon.perdrix@loria.fr

Vladimir Zamdzhiev
Université de Lorraine, CNRS, Inria,
LORIA, F-54000 Nancy
France
vladimir.zamdzhiev@inria.fr

Abstract

We introduce a new kind of expectation transformer for a mixed *classical-quantum programming language*. Our semantic approach relies on a new notion of a cost structure, which we introduce and which can be seen as a specialisation of the Kegelspitzen of Keimel and Plotkin. We show that our weakest precondition analysis is both sound and adequate with respect to the operational semantics of the language. Using the induced expectation transformer, we provide formal analysis methods for the expected cost analysis and expected value analysis of classical-quantum programs. We illustrate the usefulness of our techniques by computing the expected cost of several well-known quantum algorithms and protocols, such as coin tossing, repeat until success, entangled state preparation, and quantum walks.

Keywords: complexity analysis, quantum programming, expectation transformer, formal semantics

1 Introduction

Quantum computation is a promising and emerging computational paradigm which can efficiently solve problems considered to be intractable on classical computers [**??**]. However, the unintuitive nature of quantum mechanics poses interesting and challenging questions for the design and analysis of quantum programming languages. Indeed, the quantum program dynamics are considerably more complicated compared to the behaviour of classical probabilistic programs. Therefore, formal reasoning about quantum programs requires the development of novel methods and tools.

An important open problem is to compute the expected resource usage of quantum programs. For example, this may be used to determine: (1) the expected runtime; (2) the expected number of quantum gates; or (3) the amount of quantum resources (in an application-specific sense) required by quantum programs, etc. The difficulty of this problem, which is undecidable, requires using elaborate methods to solve it whenever possible. These methods for estimating resource usage must be compositional, systematic, and, preferably, tractable; this excludes de facto any direct use of the operational semantics.

We address this open problem by establishing a weakest precondition reasoning in the form of a *quantum expectation transformer*, named `qet`-*calculus*, that is rich enough to recover earlier wp-calculi in the context of classical programs as well as denotational semantics for quantum programs. Further, the calculus appears to be the right foundation for subsequent automation of the method, which however, is left for future work. The exact solution of the expected cost problem can be recovered via this calculus, and furthermore, our method may also be used to find *approximate* solutions by identifying suitable upper bounds. Therefore, our method provides a basis for attacking and ameliorating this undecidable problem in a systematic and compositional way.

1.1 Our Contributions

As a first step towards achieving our main objective, we introduce a new domain-theoretic notion, called a *cost structure* (Section 2). It is based on Kegelspitzen [**?**], which are dcpo's (directed-complete partial orders) equipped with a suitable convex structure that may be used to reason about the semantics of probabilistic [**??**] and quantum programming languages [**?**]. A cost structure is then a pair $(\mathsf{S}, \hat{+})$ of a

LICS '22, August 2–5, 2022, Haifa, Israel

ACM ISBN 978-1-4503-9351-5/22/08…$15.00
https://doi.org/10.1145/3531130.3533332

Kegelspitze S together with a *cost addition* operation $\hat{+}$ that allows us to model resource consumption in a coherent way.

We introduce a mixed classical-quantum programming language on which we formally define the expected cost and the expected value of programs. Our programming language (Section 3) supports conditional branching, while loops, the usual quantum primitives (including quantum measurements), classical data, and a special statement for resource consumption. To seamlessly model the combination of cost primitives and probabilistic choice — induced by quantum measurements — we define the operational semantics of our language as a probabilistic abstract reduction system [?], whose reduction rules are annotated by costs [?].

In Section 4, we introduce the aformentioned qet-calculus, which can be seen as a generalisation of previous work on predicate transformers and probabilistic expectation transformers. For a given cost structure $(\mathsf{S}, \hat{+})$, our quantum expectation transformer is a semantic function

$$\mathtt{qet}\left[\cdot\right]\left\{\cdot\right\} : \mathtt{Program} \to \mathsf{S}^{\mathsf{State}} \to \mathsf{S}^{\mathsf{State}}$$

with the informal meaning that having f resources available after execution of the program $\mathtt{stm}$ requires $\mathtt{qet}\left[\mathtt{stm}\right]\left\{f\right\}$ resources overall, in expectation. Following the literature, we will call functions $f \in \mathsf{S}^{\mathsf{State}}$, i.e. functions from quantum program states to a cost structure, *expectations*. From a different perspective, the qet-transformer can be seen as a semantics for quantum programs in continuation passing style, enabling compositional reasoning. We prove that this semantics enjoys nice algebraic and domain-theoretic properties (§4.2) and that it is sound and adequate with respect to the operational semantics (§4.3). As a consequence, we prove that the expected cost of a program in our mixed classical-quantum language (as defined via the operational semantics) is precisely recovered by using our quantum expectation transformer (Corollary 4.8). Furthermore, because our semantics is defined in a suitable level of generality, by choosing an appropriate cost structure $(\mathsf{S}, \hat{+})$, we show how a strongly adequate quantum denotational semantics may be defined as a *special case* (§4.4), which highlights important connections between our approach and denotational semantics of probabilistic and quantum programming languages.

The usefulness of our methods are illustrated through a running example that performs (unbounded) coin tossing using quantum resources. More useful and complicated quantum programs are analysed in Section 5, where we show how we can determine the expected cost of these programs using our quantum expectation transformer method.

1.2 Related Work

Classical and probabilistic programs. *Predicate transformer* semantics were introduced in the seminal works of [?] and [?] as a method for reasoning about the semantics of imperative programs. Predicate transformers map each program statement to a function between two predicates on the state space of the program. Consequently, their semantics can be viewed as a reformulation of Floyd–Hoare logic [?], since they transform postconditions (the output predicate) to preconditions (the input predicate).

This methodology has been extended to probabilistic programs by replacing predicates with expectations, leading to the notion of *expectation transformers* (see [?]) and the development of weakest pre-expectation semantics [?]. Expectation transformers have been used to reason about expected values [?], but also runtimes [?], and costs [???].

Quantum programs. The articles [?] and [?] present two first attempts to adapt expectation transformers to the runtime analysis of quantum programs. Other than using expectation transformers, we do not know of any other technique that can be used to determine the expected cost of quantum programs.

The paper [?] discusses the interest of adapting the method to the quantum case through a running example. However, no correctness results (soundness or adequacy) are proved.

Most recently, **?** defines a notion of expected runtime transformer *ert* which, on the surface, closely resembles our expectation transformer. We emphasise, however, that their notion of *ert* depends on the denotational semantics of the program. The two main distinctive features of our approach are its compositional definition, particularly our definition does not refer back to the semantics of the program, and that it operates on expectations over arbitrary cost structures, rather than on partial density operators. This has several implications. Foremost, our approach is applicable to the more general setting of mixed classical-quantum programs, almost-sure termination is not a prerequisite, and it is not constrained to reason about runtimes. Indeed, instances of our expectation transformer permit reasoning about probabilities of predicates, expectations and arbitrary (non-decreasing) costs (see Definition 4.2), such as runtimes, the number of certain gates used, etc. Moreover, by lifting *upper invariants* [?] to our setting we enable approximate reasoning and eliminate the need to reason about fixpoints or limits, stemming from the semantics of loops. As our quantum expectation transformers can be seen as a proper generalisation of expectation transformers for probabilistic programs, which have already been successfully implemented [??], we believe that our approach constitutes a good candidate for developing methods for automatic cost analysis of quantum programs.

2 Kegelspitzen and Cost Structures

We begin by defining a notion of *cost structure* based on the domain-theoretic and convex structure of Kegelspitzen. This is used in later sections by our quantum expectation transformers in order to formalise the semantics.

Kegelspitzen [?] are dcpo's (directed complete partial orders) that enjoy a convex structure. We define our quantum expectation transformer by making use of Kegelspitzen,

but for simplicity, we define Kegelspitzen using ω-cpo's (ω-complete partial orders), instead of dcpo's, because the former notion is more familiar to most readers.

Definition 2.1. An *ω-chain* in a partial order $(X, \leq)$ is a countable increasing sequence of elements of X, i.e., a sequence $(x_i)_{i\in\mathbb{N}}$, such that $x_i \leq x_j$ for any $i \leq j$. An *ω-cpo (ω-complete partial order)* is a partial order $(X, \leq)$, such that every ω-chain in X has a supremum (least upper bound) within X. A monotone function $f : X \to Y$ between two ω-cpos is *ω-continuous* if it preserves suprema of ω-chains.

Next, we recall barycentric algebras, which allows us to take convex combinations of elements in a coherent way.

Definition 2.2 ([?])**.** A *barycentric algebra* is a set A equipped with binary operations $a +_r b$, one for every real number $r \in [0, 1]$, such that for all $a, b, c \in A$ and $p \in [0, 1)$, the following equalities hold:

$$a +_1 b = a; \qquad a +_r b = b +_{1-r} a;$$
$$a +_r a = a; \qquad (a +_p b) +_r c = a +_{pr} (b +_{\frac{r-pr}{1-pr}} c).$$

Next we introduce pointed barycentric algebras, which allow us to also define scalar multiplication in a natural way.

Definition 2.3 ([?])**.** A *pointed barycentric algebra* is a barycentric algebra A equipped with a distinguished element $\bot$. For $a \in A$ and $r \in [0, 1]$, we define scalar multiplication as $r \cdot a \triangleq a +_r \bot$.

We can now define an ω-Kegelspitze as a pointed barycentric algebra that respects the order of an ω-cpo.

Definition 2.4. An *ω-Kegelspitze* is a pointed barycentric algebra K equipped with an ω-complete partial order such that, (1) scalar multiplication $(r, a) \mapsto r \cdot a \colon [0, 1] \times K \to K$ is ω-continuous in both arguments, and (2) for every $r \in [0, 1]$ the functions $(a, b) \mapsto a +_r b \colon K \times K \to K$ are ω-continuous in both arguments.

For brevity, we will refer to ω-Kegelspitzen simply as Kegelspitzen. In fact, all ω-Kegelspitzen we consider in this paper are also Kegelspitzen in the sense of [?] (i.e., as dcpo's), so this should not lead to confusion. We note that, in every Kegelspitze K, scalar multiplication $(r, a) \mapsto r \cdot a = a +_r \bot$ is ω-continuous and therefore monotone in the r-component, which implies $\bot = \bot +_1 a = a +_0 \bot = 0 \cdot a \leq 1 \cdot a = a$ for each $a \in K$. Therefore, the distinguished element $\bot$ is the least element of K.

Example 2.5. The real unit interval $[0, 1]$ is a Kegelspitze in the usual order when we define $a +_r b \triangleq ra + (1 - r)b$ and $\bot \triangleq 0$. The same assignment can also be used to equip the extended non-negative reals $\mathbb{R}^{+\infty} \triangleq \mathbb{R}^+ \cup \{\infty\}$ with the structure of a Kegelspitze. Note that the non-negative reals $\mathbb{R}^+$ is *not* a Kegelspitze, because it lacks an ω-cpo structure.

Next, we consider some Kegelspitzen which are important for the semantics of quantum programming languages.

Example 2.6. A *density matrix* is a positive semi-definite hermitian matrix A (i.e., a hermitian matrix all of whose eigenvalues are nonnegative), such that $\mathrm{tr}(A) = 1$. A *subdensity matrix* is a positive semi-definite hermitian matrix A, such that $\mathrm{tr}(A) \leq 1$. Let $D_n \subseteq \mathbb{C}^{n\times n}$ be the set of subdensity matrices of dimension n. Then D_n is an ω-cpo when equipped with the Löwner order (a generalization of the standard order on the reals to density matrices): $A \leq B$ iff $B - A$ is positive semi-definite [?]. Moreover, D_n has the structure of a Kegelspitze under the assignment $\bot \triangleq \mathbf{0}$ and $A +_r B \triangleq rA + (1 - r)B$.

Recall that density matrices are used in quantum physics to represent probabilistic mixtures of pure quantum states. In quantum programming semantics, we use *sub*density matrices in order to account for the probability of non-termination.

Kegelspitzen may also be used to define convex sums.

Definition 2.7. In a Kegelspitze K, for $a_i \in K, r_i \in [0, 1]$ with $\sum_{i=1}^n r_i \leq 1$, we define the *convex sum* inductively by:

$$\sum_{i=1}^{n} r_i a_i \triangleq \begin{cases} \bot & \text{if } n = 0, \\ a_n & \text{if } n > 0 \text{ and } r_n = 1, \\ a_n +_{r_n} \left(\sum_{i=1}^{n-1} \frac{r_i}{1-r_n} a_i\right) & \text{otherwise.} \end{cases}$$

In fact, the expression $\sum_{i=1}^n r_i a_i$ is ω-continuous in each r_i and a_i and the sum is also invariant under index permutation (see [?] for more details). Countable convex sums may be defined as follows: given $a_i \in K$ and $r_i \in [0, 1]$, for $i \in \mathbb{N}$, with $\sum_{i\in\mathbb{N}} r_i \leq 1$, let $\sum_{i\in\mathbb{N}} r_i a_i \triangleq \sup_{n\in\mathbb{N}} \sum_{j=1}^n r_j a_j$.

We now formalize a notion of *cost structure* for expectation transformers in the context of quantum programs. This can be seen as a Kegelspitze equipped with an operation for injecting a cost — modeled as a positive real number — into the Kegelspitze, which satisfies some coherence conditions with respect to the structure of the Kegelspitze.

Definition 2.8. A *cost structure* $\mathcal{S} = (\mathsf{S}, \hat{+})$ is a Kegelspitze S equipped with an operation $\hat{+} : \mathbb{R}^{+\infty} \times \mathsf{S} \to \mathsf{S}$ that is ω-continuous in both arguments and satisfies the identities

$$0 \mathbin{\hat{+}} s = s \tag{1}$$
$$c_1 \mathbin{\hat{+}} (c_2 \mathbin{\hat{+}} s) = (c_1 + c_2) \mathbin{\hat{+}} s \tag{2}$$
$$(c_1 \mathbin{\hat{+}} s_1) +_r (c_2 \mathbin{\hat{+}} s_2) = (c_1 +_r c_2) \mathbin{\hat{+}} (s_1 +_r s_2) \tag{3}$$

Example 2.9. For any Kegelspitze S, we get a cost structure $(\mathsf{S}, +_\mathsf{f})$ with *forgetful cost addition* defined by $c +_\mathsf{f} r \triangleq r$. A more representative example is given by the cost structure $(\mathbb{R}^{+\infty}, +)$, where $+$ is the standard addition in $\mathbb{R}^{+\infty}$.

3 Quantum Programming Language

In this section we introduce the syntax and operational semantics of our imperative programming language supporting both quantum and classical programming primitives.

3.1 Syntax

Let $\mathcal{B}$, $\mathcal{V}$ and $\mathcal{Q}$ be three distinct types for Boolean, numerical, and qubit data. We will use variables $\mathtt{x}, \mathtt{y}, \mathtt{z}$ to range over classical variables of type $\mathcal{K} \in \{\mathcal{B}, \mathcal{V}\}$ and we will use $\mathtt{q}, \mathtt{q}_1, \mathtt{q}_2$, etc., to range over quantum variables of type $\mathcal{Q}$. The syntax of quantum programs is described in Figure 1, where n is a constant in $\mathbb{Z}$, $\mathtt{U}$ is an operator symbol of arity $ar(\mathtt{U}) \in \mathbb{N} - \{0\}$, $\overline{\mathtt{q}}$ stands for a sequence of qubit variables $\mathtt{q}_1, \ldots, \mathtt{q}_{ar(\mathtt{U})}$, and $\mathtt{meas}(\mathtt{q})$ represents the standard measurement on qubit $\mathtt{q}$ in the computational basis: the outcome of a measurement will be the Boolean value 0 or 1. When needed, variables and expressions can be annotated by their type as superscript. If $\mathtt{a}$ evaluates to a positive integer c, the statement $\mathtt{consume}(\mathtt{a})$ consumes c resource units but acts as a no-op otherwise. That is, we permit only non-negative costs. This restriction is in place to ensure that the notion of expected cost — to be defined in a moment — is well-defined.

Program variables are global. For a given expression or statement t, let $\mathcal{B}(t)$ (respectively $\mathcal{V}(t)$, $\mathcal{Q}(t)$) be the set of Boolean (resp. numerical, qubit) variables in t.

Example 3.1. Let $\mathtt{H}$ be the operator symbol representing the Hadamard unitary operation. The program $CT(\mathtt{q})$ in Listing 1 performs coin tossing by repeatedly measuring an initial qubit $\mathtt{q}$ (which may be mapped into a superposition state via $\mathtt{H}$) until the measurement outcome $\mathtt{false}$ occurs. This program will be our simple running example throughout the paper. Its probability to terminate within n steps depends on the initial state of the qubit $\mathtt{q}$ and the loop consumes 1 resource for each iteration. The overall probability of termination (in any number of steps) is 1.

3.2 Operational Semantics

In what follows, we model the dynamics of our language as a probabilistic abstract reduction system [?] — a transition system where reducts are chosen from a probability distribution. Reductions can then be defined as stochastic processes [?], or equivalently, as reduction relations over distributions [?]. We follow the latter approach; unlike the former it permits us to define a notion of expected cost concisely, without much technical overhead [?].

Probabilistic Abstract Reduction Systems (PARS). Let A be a set of *objects*. A discrete subdistribution δ over A is a function $\delta : A \to [0,1]$ with countable support that maps an element a of A to a probability $\delta(a)$ such that $\sum_{a \in \mathrm{supp}(\delta)} \delta(a) \leq 1$. If $\sum_{a \in \mathrm{supp}(\delta)} \delta(a) = 1$ then δ is a discrete distribution. We only consider discrete (sub)distributions and we shall simply refer to them as (sub)distributions from now on. Any (sub)distribution δ can be written as $\{\delta(a) : a\}_{a \in \mathrm{supp}(\delta)}$. The set of subdistributions over A is denoted by $\mathcal{D}(A)$. Note that $\mathcal{D}(A)$ is closed under convex combinations $\sum_i p_i \cdot \delta_i \triangleq \lambda a. \sum_i p_i \delta_i(a)$ for countably many probabilities $p_i \in [0,1]$ such that $\sum_i p_i \leq 1$. The notion of expectation of a function $f : A \to S$, where S is a Kegelspitze, is defined for a given subdistribution δ over A by $\mathbb{E}_\delta(f) \triangleq \Sigma_{a \in \mathrm{supp}(\delta)} \delta(a) \cdot f(a)$.

A *(weighted) Probabilistic Abstract Reduction System* (PARS) on A is a ternary relation $\cdot \xrightarrow{\cdot} \cdot \subseteq A \times \mathbb{R}^+ \times \mathcal{D}(A)$. For $a \in A$, a rule $a \xrightarrow{c} \{\delta(b) : b\}_{b \in A}$ indicates that a reduces to b with probability $\delta(b)$ and cost $c \in \mathbb{R}^+$. Given two objects a and b, $a \xrightarrow{c} \{1 : b\}$ will be written $a \xrightarrow{c} b$ for brevity. For simplicity, we consider only deterministic PARSs $\to$, i.e., $a \xrightarrow{c_1} \delta_1$ and $a \xrightarrow{c_2} \delta_2$ implies $c_1 = c_2$ and $\delta_1 = \delta_2$. An object $a \in A$ is called terminal if there is no rule $a \xrightarrow{c} \delta$, which we write as $a \not\to$.

Every deterministic PARS $\to$ over A can be lifted to a ternary weighted reduction relation $\cdot \xrightarrow{\cdot}\!\!\!\!\!\rightarrow \cdot \subseteq \mathcal{D}(A) \times \mathbb{R}^+ \times \mathcal{D}(A)$ in a natural way, see Figure 2. A reduction step $\delta \xrightarrow{c}\!\!\!\!\!\rightarrow \epsilon$ indicates that the subdistribution of objects δ evolves to a subdistribution of reducts ϵ in one step, with an expected cost of c. Note that since $\to$ is deterministic, so is the reduction relation $\xrightarrow{\cdot}\!\!\!\!\!\rightarrow$. We denote by $\delta \xrightarrow{c}\!\!\!\!\!\rightarrow_n \epsilon$ the n-fold ($n \geq 0$) composition of $\xrightarrow{\cdot}\!\!\!\!\!\rightarrow$ with expected cost c, defined by $\delta \xrightarrow{c}\!\!\!\!\!\rightarrow_n \epsilon$ if $\delta \xrightarrow{c_1}\!\!\!\!\!\rightarrow \cdots \xrightarrow{c_n}\!\!\!\!\!\rightarrow \epsilon$ and $c = \sum_{i=1}^n c_i$. In particular, $\delta \xrightarrow{0}\!\!\!\!\!\rightarrow_0 \delta$.

Let us illustrate these notions on a small example.

Example 3.2. We fix objects $A = \mathbb{Z} \cup \{\mathrm{geo}(n) \mid n \in \mathbb{Z}\}$. Consider the PARS $\to_{\mathrm{geo}}$ over A defined through the rules

$$\mathrm{geo}(n) \xrightarrow{1}_{\mathrm{geo}} \{1/2 : n+1, 1/2 : \mathrm{geo}(n+1)\} \qquad (n \in \mathbb{Z}),$$

stating that $\mathrm{geo}(n)$ increments its argument and then either returns or recurs, in each case with probability $1/2$ and cost one. Starting from $\mathrm{geo}(0)$, this PARS admits precisely one infinite reduction sequence

$$\begin{aligned}\{1 : \mathrm{geo}(0)\} &\xrightarrow{1}\!\!\!\!\!\rightarrow_{\mathrm{geo}} \{1/2 : 1, 1/2 : \mathrm{geo}(1)\} \\ &\xrightarrow{1/2}\!\!\!\!\!\rightarrow_{\mathrm{geo}} \{1/2 : 1, 1/4 : 2, 1/4 : \mathrm{geo}(2)\} \\ &\xrightarrow{1/4}\!\!\!\!\!\rightarrow_{\mathrm{geo}} \{1/2 : 1, 1/4 : 2, 1/8 : 3, 1/8 : \mathrm{geo}(3)\} \\ &\xrightarrow{1/8}\!\!\!\!\!\rightarrow_{\mathrm{geo}} \cdots\end{aligned}$$

This sequence approaches the distribution $\{1/2^n : n\}_{n>0}$ of terminal objects in $\mathbb{Z}$, with an expected cost of $\sum_{i=0}^{\infty} 1/2^i = 2$.

As indicated in this example, for every $\delta \in \mathcal{D}(A)$ there is precisely one infinite sequence $\delta = \delta_0 \xrightarrow{c_0}\!\!\!\!\!\rightarrow \delta_1 \xrightarrow{c_1}\!\!\!\!\!\rightarrow \delta_2 \xrightarrow{c_2}\!\!\!\!\!\rightarrow \cdots$ gradually approaching a *normal form distribution* of terminal objects with an expected cost of $\sum_{i=0}^{\infty} c_i$.[1] Note that this normal form distribution can be a proper subdistribution — in which case the PARS is not almost-surely terminating — and that the cost can be infinite.

[1] This infinite sum is always defined, since costs c_i are non-negative.

$$
\begin{array}{llll}
\texttt{AExp} & \mathsf{a}, \mathsf{a}_1, \mathsf{a}_2 & ::= & \mathsf{x}^{\mathcal{V}} \mid n \mid \mathsf{a}_1 + \mathsf{a}_2 \mid \mathsf{a}_1 - \mathsf{a}_2 \mid \mathsf{a}_1 \times \mathsf{a}_2 \\
\texttt{BExp} & \mathsf{b}, \mathsf{b}_1, \mathsf{b}_2 & ::= & \mathsf{x}^{\mathcal{B}} \mid \texttt{true} \mid \texttt{false} \mid \mathsf{a}_1 = \mathsf{a}_2 \mid \mathsf{a}_1 \leq \mathsf{a}_2 \mid \mathsf{a}_1 < \mathsf{a}_2 \mid \neg \mathsf{b} \mid \mathsf{b}_1 \wedge \mathsf{b}_2 \mid \mathsf{b}_1 \vee \mathsf{b}_2 \\
\texttt{Exp} & \mathsf{e}, \mathsf{e}_1, \mathsf{e}_2 & ::= & \mathsf{a} \mid \mathsf{b} \\
\texttt{Statement} & \texttt{stm}, \texttt{stm}_1, \texttt{stm}_2 & ::= & \texttt{skip} \mid \mathsf{x}^{\mathcal{K}} = \mathsf{e}^{\mathcal{K}} \mid \overline{\mathsf{q}} \mathrel{*}= \mathsf{U} \mid \mathsf{x}^{\mathcal{B}} = \texttt{meas(q)} \mid \texttt{consume(a)} \\
& & & \mid \texttt{stm}_1; \texttt{stm}_2 \mid \texttt{if(b)\{stm}_1\texttt{\} else \{stm}_2\texttt{\}} \mid \texttt{while(b)\{stm\}}
\end{array}
$$

Figure 1. Syntax of quantum programs.

```
CT(q) ≜ x = true;
        while(x){
          q *= H;
          x = meas(q);   } stm0   } stm
          consume(1)
        }
```

Listing 1. Coin tossing.

$$
\frac{a \not\rightarrow}{\{1:a\} \overset{0}{\twoheadrightarrow} \{1:a\}}\ (\text{Term}) \qquad \frac{a \xrightarrow{c} \delta}{\{1:a\} \overset{c}{\twoheadrightarrow} \delta}\ (\text{Mono})
$$

$$
\frac{\delta_i \overset{c_i}{\twoheadrightarrow} \epsilon_i \qquad \sum_i p_i \leq 1}{\sum_i p_i \cdot \delta_i \overset{\sum_i p_i c_i}{\twoheadrightarrow} \sum_i p_i \cdot \epsilon_i}\ (\text{Muti})
$$

Figure 2. Weigthed reduction relation induced by PARS.

Based on these intuitions, for an object $a \in A$, we define the expected cost function $\mathrm{ecost}_{\rightarrow} : A \to \mathbb{R}^{+\infty}$ by

$$\mathrm{ecost}_{\rightarrow}(a) \triangleq \sup_{n \in \mathbb{N}} \{c \mid \{1:a\} \overset{c}{\twoheadrightarrow}_n \delta\},$$

and the normal form function $\mathsf{nf}_{\rightarrow} : A \to \mathcal{D}(A)$ by

$$\mathsf{nf}_{\rightarrow}(a) \triangleq \sup_{n \in \mathbb{N}} \{\delta\restriction_{term} \mid \{1:a\} \overset{c}{\twoheadrightarrow}_n \delta\},$$

where $\delta\restriction_{term}$ is the restriction of δ to terminal objects, i.e., $\delta\restriction_{term} \triangleq \{\delta(a) : a \mid a \not\rightarrow\}_{a \in \mathrm{supp}(\delta)}$, and the supremum is taken w.r.t. the pointwise order of subdistributions. Note that $\mathsf{nf}_{\rightarrow}(a)$ is well-defined, which essentially follows from the fact that $(\delta_n\restriction_{term})_{n \in \mathbb{N}}$, for δ_n such that for $\{1:a\} \overset{c}{\twoheadrightarrow}_n \delta_n$, is a monotonically increasing sequence, by definition of $\overset{c}{\twoheadrightarrow}$.

Quantum Programs as PARSs. We now endow quantum programs with an operational semantics defined through a PARS, operating on pairs of classical and quantum states.

Let $\mathbb{C}$ denote the set of complex numbers. Given a set Q of n qubit variables, let $\mathcal{H}_Q$ be the Hilbert space $\mathbb{C}^{2^n}$ of n qubits[2]. We use Dirac notation, $|\varphi\rangle$, to denote a quantum state of $\mathcal{H}_Q$. Any quantum state $|\varphi\rangle$ can be written as $\Sigma_{b \in \{0,1\}^n} \alpha_b |b\rangle$, with $\alpha_b \in \mathbb{C}$, and $\Sigma_{b \in \{0,1\}^n} |\alpha_b|^2 = 1$. $\langle\varphi|$ is the conjugate transpose of $|\varphi\rangle$, i.e., $\langle\varphi| \triangleq |\varphi\rangle^{\dagger}$. $\langle\varphi|\psi\rangle \triangleq \langle\varphi|\,|\psi\rangle$ and $|\varphi\rangle\langle\psi|$ denote the inner product and outer product of $|\varphi\rangle$ and $|\psi\rangle$, respectively. The norm of a vector is defined by $\||\varphi\rangle\| \triangleq \sqrt{\langle\varphi|\varphi\rangle}$. We define (linear) operators over $\mathcal{H}_Q$ as linear maps. Hence an operator will be represented by a square matrix whose dimension is equal to the dimension of $\mathcal{H}_Q$. Given $m \geq 1$, let I_m be the $m \times m$ identity matrix and $\otimes$ be the standard Kronecker product on matrices.

Assume that $Q = \{\mathsf{q}_1, \ldots, \mathsf{q}_n\}$. For $k \in \{0,1\}$, let $|k\rangle_{\mathsf{q}_i} \in \mathcal{H}_Q$ be defined by $|k\rangle_{\mathsf{q}_i} \triangleq I_{2^{i-1}} \otimes |k\rangle \otimes I_{2^{n-i}}$ and let $\langle k|_{\mathsf{q}_i}$ be its conjugate transpose. The measurement of a qubit $\mathsf{q} \in Q$ of a state $|\varphi\rangle \in \mathcal{H}_Q$ produces the classical outcome $k \in \{0,1\}$ with probability $p_k^{\mathsf{q}}|\varphi\rangle$, and transforms the quantum state $|\varphi\rangle$ into $\mathtt{M}_k^{\mathsf{q}}|\varphi\rangle$, where $\mathtt{M}_k^{\mathsf{q}} : \mathcal{H}_Q \to \mathcal{H}_Q$ is defined as

$$\mathtt{M}_k^{\mathsf{q}} \triangleq |\varphi\rangle \mapsto \frac{|k\rangle_{\mathsf{q}} \langle k|_{\mathsf{q}} |\varphi\rangle}{\left\| \langle k|_{\mathsf{q}} |\varphi\rangle \right\|}$$

and $p_k^{\mathsf{q}} : \mathcal{H}_Q \to [0,1]$ is defined as $p_k^{\mathsf{q}} \triangleq |\varphi\rangle \mapsto \left\| \langle k|_{\mathsf{q}} |\varphi\rangle \right\|^2$.

The classical state is modelled as a (well-typed) *store* s. For two given sets B and V of Boolean and numerical variables, a (classical) *store* s is a pair of maps (s^B, s^V) such that $s^B : B \to \{0,1\}$ and $s^V : V \to \mathbb{Z}$. The domain of s, noted $dom(s)$, is defined by $dom(s) \triangleq B \cup V$. Given a store $s = (s^B, s^V)$, we let $s[\mathsf{x}^{\mathcal{V}} := k]$ (resp. $s[\mathsf{x}^{\mathcal{B}} := k], k \in \{0,1\}$) be the store obtained from s by updating the value assigned to x in the map s^V (resp. s^B) to k. Define also $s(\mathsf{x}^{\mathcal{V}}) \triangleq s^V(\mathsf{x}^{\mathcal{V}})$ and $s(\mathsf{x}^{\mathcal{B}}) \triangleq s^B(\mathsf{x}^{\mathcal{B}})$. Given a store s, let $[\![-]\!]^s$ be the map associating to each expression e (and such that $\mathcal{B}(\mathsf{e}) \cup \mathcal{V}(\mathsf{e}) \subseteq dom(s)$) of type $\mathcal{V}$, a value in $\mathbb{Z}$, and to each expression e of type $\mathcal{B}$ a value in $\{0,1\}$, and defined in a standard way. For example $[\![\mathsf{x}]\!]^s \triangleq s(\mathsf{x})$, $[\![n]\!]^s \triangleq n$, $[\![\texttt{true}]\!]^s \triangleq 1$, etc.

A *state* σ is a pair $(s, |\varphi\rangle)$ consisting of a store s and a quantum state $|\varphi\rangle$. A *configuration* μ for statement $\texttt{stm}$ has the form $(\texttt{stm}, \sigma)$, sometimes written as $(\texttt{stm}, s, |\varphi\rangle)$ for $\sigma = (s, |\varphi\rangle)$. Let $\texttt{State}$ and $\texttt{Conf}$ be the set of states and the set of configurations, respectively. A configuration $(\texttt{stm}, s, |\varphi\rangle)$ is well-formed with respect to the sets of variables B, V, Q if $\mathcal{B}(\texttt{stm}) \subseteq B$, $\mathcal{V}(\texttt{stm}) \subseteq V$, $\mathcal{Q}(\texttt{stm}) \subseteq Q$, $dom(s) = B \cup V$, and $|\varphi\rangle \in \mathcal{H}_Q$. Throughout the paper, we only consider configurations that are well-formed with respect to the sets of variables of the program under consideration.

The operational semantics is described in Figure 3 as a PARS $\rightarrow$ over objects in $\texttt{Conf} \cup \texttt{State}$, where precisely the

[2] We assume Q to be a totally ordered set so that the smallest element of Q corresponds to the first qubit of $\mathcal{H}_Q$ and so on.

$$\frac{}{(\mathtt{skip}, s, |\varphi\rangle) \xrightarrow{0} (s, |\varphi\rangle)}\ (\text{Skip}) \qquad \frac{}{(\mathtt{x} = \mathtt{e}, s, |\varphi\rangle) \xrightarrow{0} (s[\mathtt{x} := [\![\mathtt{e}]\!]^s], |\varphi\rangle)}\ (\text{Exp}) \qquad \frac{}{(\overline{\mathtt{q}} *= \mathtt{U}, s, |\varphi\rangle) \xrightarrow{0} (s, \mathtt{U}_{\overline{\mathtt{q}}} |\varphi\rangle)}\ (\text{Op})$$

$$\frac{}{(\mathtt{x} = \mathtt{meas}(\mathtt{q}), s, |\varphi\rangle) \xrightarrow{0} \{p_k^{\mathtt{q}} |\varphi\rangle : (s[\mathtt{x} := k], \mathtt{M}_k^{\mathtt{q}} |\varphi\rangle)\}_{k\in\{0,1\}}}\ (\text{Meas}) \qquad \frac{}{(\mathtt{consume}(\mathtt{a}), s, |\varphi\rangle) \xrightarrow{\max([\![\mathtt{a}]\!]^s, 0)} (s, |\varphi\rangle)}\ (\text{Cons})$$

$$\frac{(\mathtt{stm}_1, s, |\varphi\rangle) \xrightarrow{c} \{p_i : (\mathtt{stm}_1^i, s^i, |\varphi^i\rangle)\}_{i\in I} \cup \{q_j : (s^j, |\varphi^j\rangle)\}_{j\in J}}{(\mathtt{stm}_1; \mathtt{stm}_2, s, |\varphi\rangle) \xrightarrow{c} \{p_i : (\mathtt{stm}_1^i; \mathtt{stm}_2, s^i, |\varphi^i\rangle)\}_{i\in I} \cup \{q_j : (\mathtt{stm}_2, s^j, |\varphi^j\rangle)\}_{j\in J}}\ (\text{Seq})$$

$$\frac{[\![\mathtt{b}]\!]^s \in \{0, 1\}}{(\mathtt{if(b)\{stm_1\}\ else\ \{stm_0\}}, s, |\varphi\rangle) \xrightarrow{0} (\mathtt{stm}_{[\![\mathtt{b}]\!]^s}, s, |\varphi\rangle)}\ (\text{Cond})$$

$$\frac{[\![\mathtt{b}]\!]^s = 0}{(\mathtt{while(b)\{stm\}}, s, |\varphi\rangle) \xrightarrow{0} (s, |\varphi\rangle)}\ (\text{Wh}_0) \qquad \frac{[\![\mathtt{b}]\!]^s = 1}{(\mathtt{while(b)\{stm\}}, s, |\varphi\rangle) \xrightarrow{0} (\mathtt{stm; while(b)\{stm\}}, s, |\varphi\rangle)}\ (\text{Wh}_1)$$

Figure 3. Operational semantics in terms of PARS.

objects in State are terminal. Rule (Cons) evaluates the arithmetic expression provided as argument to a cost, an integer, and annotates the reduction with this cost, whenever it is a positive integer (otherwise the cost is 0). The state of a configuration can only be updated by the three rules (Exp), (Op), and (Meas). Rule (Exp) updates the classical store in a standard way. Rule (Op) updates the quantum state to a new quantum state $\mathtt{U}_{\overline{\mathtt{q}}} |\varphi\rangle$, where $\mathtt{U}_{\overline{\mathtt{q}}}$ is the map that applies the unitary operator U to qubits in $\overline{\mathtt{q}} = \mathtt{q}_1, \ldots, \mathtt{q}_{ar(\mathtt{U})}$ and tensoring the map with the identity on all other qubits to match the dimension of $|\varphi\rangle$. Rule (Meas) performs a measurement on qubit q. This rule returns a distribution of configurations corresponding to the two possible outcomes, $k = 0$ and $k = 1$, with their respective probabilities $p_k^{\mathtt{q}} |\varphi\rangle$ and, in each case, updates the classical store and the quantum state accordingly. Rule (Seq) governs the execution of a sequence of statements $\mathtt{stm}_1; \mathtt{stm}_2$. The rule accounts for potential probabilistic behavior when $\mathtt{stm}_1$ performs a measurement and it is otherwise standard. All the other rules are standard.

For a statement stm, we overload the notion of expected cost function and define $\mathrm{ecost}_{\mathtt{stm}} : (\mathbb{R}^{+\infty})^{\mathsf{State}}$ by

$$\mathrm{ecost}_{\mathtt{stm}}(\sigma) \triangleq \mathrm{ecost}_{\to}(\mathtt{stm}, \sigma).$$

Moreover, the function $\mathrm{evalue}_{\mathtt{stm}} : \mathsf{S}^{\mathsf{State}} \to \mathsf{S}^{\mathsf{State}}$ defined by

$$\mathrm{evalue}_{\mathtt{stm}}(f)(\sigma) \triangleq \mathbb{E}_{\mathsf{nf}_{\to}(\mathtt{stm},\sigma)}(f)$$

gives the expected value of f on the subdistribution of terminal states obtained by executing stm on state σ. Note that this function is well-defined, as $\mathsf{nf}_{\to}(\mathtt{stm}, \sigma)$ is a sub-distribution over State.

Example 3.3. Consider the program from Example 3.1. Let stm refer to the while loop. On a state $(s, |\varphi\rangle)$ such that $|\varphi\rangle = \alpha |0\rangle + \beta |1\rangle$ (with $|\alpha|^2 + |\beta|^2 = 1$) and $s(\mathtt{x}) = 1$, it holds that:

$$\begin{aligned}
&\delta_0 \triangleq \{1 : (\mathtt{stm}, s, |\varphi\rangle)\} \\
&\xrightarrow{0}\!\!\!\!\to \{1 : (\mathtt{q} *= \mathtt{H}; \mathtt{x} = \mathtt{meas}(\mathtt{q}); \mathtt{consume}(1); \mathtt{stm}, s, |\varphi\rangle)\} && (4)\\
&\xrightarrow{0}\!\!\!\!\to \{1 : (\mathtt{x} = \mathtt{meas}(\mathtt{q}); \mathtt{consume}(1); \mathtt{stm}, s, \mathtt{H}|\varphi\rangle)\} && (5)\\
&\xrightarrow{0}\!\!\!\!\to \{p_k : (\mathtt{consume}(1); \mathtt{stm}, s[\mathtt{x} := k], |k\rangle)\}_{k\in\{0,1\}}, && (6)
\end{aligned}$$

with $p_0 = \frac{|\alpha+\beta|^2}{2}$, and $p_1 = \frac{|\alpha-\beta|^2}{2}$. The above reductions are obtained by applying rules of Figure 3 together with rule (Mono) of Figure 2: (Wh_1) for (4); (Op) and (Seq) for (5); (Meas) and (Seq) for (6).

Moreover, by rules (Cons), (Seq) , and (Mono), $\forall k \in \{0, 1\}$,

$$\{1 : (\mathtt{consume}(1); \mathtt{stm}, s[\mathtt{x} := k], |k\rangle)\} \xrightarrow{1}\!\!\!\!\to \{1 : (\mathtt{stm}, s[\mathtt{x} := k], |k\rangle)\}. \tag{7}$$

Consequently, using (4)-(7) and rule (Multi) of Figure 2:

$$\{1 : (\mathtt{stm}, s, |\varphi\rangle)\} \xrightarrow{1}\!\!\!\!\to_4 \{p_k : (\mathtt{stm}, s[\mathtt{x} := k], |k\rangle)\}_{k\in\{0,1\}},$$

as $p_0 + p_1 = 1$. Iterating the above reduction, it holds that

$$\{1 : (\mathtt{stm}, s, |1\rangle)\} \xrightarrow{1}\!\!\!\!\to_4 \{1/2 : (\mathtt{stm}, s[\mathtt{x} := k], |k\rangle)\}_{k\in\{0,1\}}$$

Moreover, as

$$\{1 : (\mathtt{stm}, s[\mathtt{x} := 0], |0\rangle)\} \xrightarrow{0}\!\!\!\!\to \{1 : (s[\mathtt{x} := 0], |0\rangle)\},$$

it holds that

$$\begin{aligned}
\delta_0 &\xrightarrow{1}\!\!\!\!\to_4 \{p_0 : (\mathtt{stm}, s[\mathtt{x} := 0], |0\rangle), p_1 : (\mathtt{stm}, s, |1\rangle)\}\\
&\xrightarrow{p_1}\!\!\!\!\to_4 \{p_0 + p_1/2 : (s[\mathtt{x} := 0], |0\rangle), p_1/2 : (\mathtt{stm}, s, |1\rangle)\}\\
&\xrightarrow{\frac{p_1}{2}}\!\!\!\!\to_4 \{p_0 + p_1/2 + p_1/4 : (s[\mathtt{x} := 0], |0\rangle), p_1/4 : (\mathtt{stm}, s, |1\rangle)\}\\
&\xrightarrow{\frac{p_1}{4}}\!\!\!\!\to_4 \cdots
\end{aligned}$$

The expected cost and the normal form are obtained as follows, by reasoning about the asymptotic behaviour.

$$\mathrm{ecost}_{CT(\mathsf{q})}(s, \begin{pmatrix}\alpha\\ \beta\end{pmatrix}) = \sup_{n\in\mathbb{N}} \left\{1 + p_1 \textstyle\sum_{i=0}^{n} \frac{1}{2^i}\right\} = 1 + |\alpha - \beta|^2$$

$$\begin{aligned}\mathrm{nf}_{CT(\mathsf{q})}(s, |\varphi\rangle) &= \sup_{n\in\mathbb{N}} \left\{\{p_0 + p_1 \textstyle\sum_{i=1}^{n} \frac{1}{2^i} : (s[\mathsf{x} := 0], |0\rangle)\}\right\}\\ &= \{1 : (s[\mathsf{x} := 0], |0\rangle)\}\end{aligned}$$

Hence, $\mathrm{evalue}_{CT(\mathsf{q})}(f)(s, |\varphi\rangle)$, the expected value of f after executing $CT(\mathsf{q})$ on $(s, |\varphi\rangle)$, is equal to $f(s[\mathsf{x} := 0], |0\rangle)$.

4 Quantum Expectation Transformers

We now revisit the expectation transformer approach for the quantum programming language introduced in Section 3. *Expectations* will be functions from the set of (classical and quantum) memory states to cost structures, i.e., functions in $\mathsf{S}^{\mathsf{State}}$, for a given cost structure S. The *quantum expectation transformer* $\mathtt{qet}[\cdot]\{\cdot\}$ is then defined in terms of a program semantics mapping expectations to expectations in a continuation passing style. Specializing the cost structure yields several quantum expectation transformers such as the *quantum expected value transformer* $\mathtt{qev}_{\mathsf{S}}[\cdot]\{\cdot\}$ and the *quantum expected cost transformer* $\mathtt{qect}[\cdot]\{\cdot\}$. After exhibiting several laws and properties of these transformers, we show their soundness and their adequacy.

4.1 Definition

Before defining expectation transformers, we introduce some preliminary notations in order to lighten the presentation.

Notations. For any expression e, $[\![\mathsf{e}]\!]$ is a shorthand notation for the function $\lambda(s, |\varphi\rangle).\, [\![\mathsf{e}]\!]^s \in (\mathbb{R}^{+\infty})^{\mathsf{State}}$ and, for any $c \in \mathbb{R}^{+\infty}$, $\underline{c}$ is the function in $(\mathbb{R}^{+\infty})^{\mathsf{State}}$ defined by $\underline{c} \triangleq \lambda\sigma.c$. To avoid notational overhead, we frequently use point-wise extensions of operations on $\hat{+}$ and $\mathbb{R}^{+\infty}$ to functions. E.g., for $p \in [0,1]^{\mathsf{State}}$, $f, g \in \mathsf{S}^{\mathsf{State}}$, $f +_p g$ denotes the function $\lambda\sigma.f(\sigma) +_{p(\sigma)} g(\sigma)$.

We will also use $f[\mathsf{x} := \mathsf{e}]$ for the expectation mapping $(s, |\varphi\rangle)$ to $f(s[\mathsf{x} := [\![\mathsf{e}]\!]^s], |\varphi\rangle)$, and similarly, for a given function $\mathsf{M} : \mathcal{H}_Q \to \mathcal{H}_Q$, $f[\mathsf{M}]$ maps $(s, |\varphi\rangle)$ to $f(s, \mathsf{M}|\varphi\rangle)$. Finally, $f[\mathsf{x} := \mathsf{e}; \mathsf{M}]$ stands for $(f[\mathsf{x} := \mathsf{e}])[\mathsf{M}]$.

Definition 4.1. Let $(\mathsf{S}, \hat{+})$ be a cost structure. The quantum expectation transformer

$$\mathtt{qet}[\cdot]\{\cdot\} : \mathtt{Program} \to \mathsf{S}^{\mathsf{State}} \to \mathsf{S}^{\mathsf{State}}$$

is defined inductively in Figure 4.

Definition 4.2 (Quantum expectation transformers instances)**.**

1. Taking the cost structure $([0,1], +_{\mathsf{f}})$ yields a *weakest precondition transformer*
$$\mathtt{qwp}[\cdot]\{\cdot\} : \mathtt{Program} \to [0,1]^{\mathsf{State}} \to [0,1]^{\mathsf{State}},$$
for probabilistic pre-condition reasoning.
2. Taking the cost structure $(\mathsf{S}, +_{\mathsf{f}})$, for any Kegelspitze S, yields an *expected value transformer*
$$\mathtt{qev}_{\mathsf{S}}[\cdot]\{\cdot\} : \mathtt{Program} \to \mathsf{S}^{\mathsf{State}} \to \mathsf{S}^{\mathsf{State}}.$$
3. Taking the cost structure $(\mathbb{R}^{+\infty}, +)$ yields an *expected cost transformer*
$$\mathtt{qect}[\cdot]\{\cdot\} : \mathtt{Program} \to (\mathbb{R}^{+\infty})^{\mathsf{State}} \to (\mathbb{R}^{+\infty})^{\mathsf{State}}.$$

4.2 Properties

The quantum expectation transformer satisfies several useful laws (Figure 5) and these laws are comparable to those found in [?].

Theorem 4.3. *All universal laws listed in Figure 5 hold.*

The (*monotonicity*) Law permits us to reason modulo upper-bounds: actual costs can be always substituted by upper-bounds. It is in fact an immediate consequence from the (*continuity*) Law, itself essential for the well-definedness of the transformer on while loops. The (*linearity*) Law is a direct consequence of the laws on cost structures. The (*upper invariant*) Law generalises the corresponding law by [?], itself a generalisation of the notion of invariant stemming from Hoare calculus from predicates to cost functions. It constitutes a complete proof rule for finding closed form upper-bounds for loops, and based on the observation that any prefix-point — as given with g in the pre-condition — is an upper bound to the least-prefixed point of a functional — in our case the expected cost of the loop (w.r.t. f).

Example 4.4. Let us search for a cost expectation of the program of Example 3.1. Recall that $\mathtt{stm}_0$ is the body of the while loop statement and that the considered cost structure is $(\mathbb{R}^{+\infty}, +)$. By (*upper invariant*) Law (see Figure 5), it suffices to find an expectation g satisfying the following inequalities

$$[\![\neg\mathsf{x}]\!] \cdot \underline{0} \le g \tag{8}$$

$$[\![\mathsf{x}]\!] \cdot \mathtt{qect}[\mathtt{stm}_0]\{g\} \le g \tag{9}$$

in order to compute an upper bound on the expectation $\mathtt{qect}[\mathtt{stm}]\{\underline{0}\}$ of the while loop statement $\mathtt{stm}$.

Using rules of Figure 4, the following equalities hold, as can be verified directly:

$$\begin{aligned}g_1 &\triangleq \mathtt{qect}[\mathtt{consume}(1)]\{g\} = \underline{1} + g\\ g_2 &\triangleq \mathtt{qect}[\mathsf{x} = \mathtt{meas}(\mathsf{q})]\{g_1\}\\ &= g_1[\mathsf{x} := 0; \mathsf{M}_0^{\mathsf{q}}] +_{p_0^{\mathsf{q}}} g_1[\mathsf{x} := 1; \mathsf{M}_1^{\mathsf{q}}]\\ &= \lambda(s, |\varphi\rangle).\, \textstyle\sum_{k\in\{0,1\}} p_k^{\mathsf{q}}(|\varphi\rangle) g_1(s[\mathsf{x} := k], |k\rangle))\\ g_3 &\triangleq \mathtt{qect}[\mathsf{q} *= \mathsf{H}]\{g_2\} = g_2[\mathsf{H}]\\ &= \lambda(s, |\varphi\rangle).\, \textstyle\sum_{k\in\{0,1\}} p_k^{\mathsf{q}}(\mathsf{H}|\varphi\rangle)(\underline{1} + g)(s[\mathsf{x} := k], |k\rangle).\end{aligned}$$

Now, we set

$$g(s, \begin{pmatrix}\alpha\\ \beta\end{pmatrix}) \triangleq [\![\mathsf{x}]\!] \cdot (1 + |\alpha - \beta|^2).$$

$$\mathtt{qet}\big[\epsilon\big]\{f\} \triangleq f \qquad \mathtt{qet}\big[\mathtt{x = meas(q)}\big]\{f\} \triangleq f[\mathtt{x} := 0; \mathtt{M}_0^{\mathtt{q}}] +_{p_0^{\mathtt{q}}} f[\mathtt{x} := 1; \mathtt{M}_1^{\mathtt{q}}],\ \text{with } p_0^{\mathtt{q}} |\varphi\rangle = \big\| \langle 0|_{\mathtt{q}} |\varphi\rangle \big\|^2$$

$$\mathtt{qet}\big[\mathtt{skip}\big]\{f\} \triangleq f \qquad \mathtt{qet}\big[\mathtt{consume(a)}\big]\{f\} \triangleq \max([\![\mathtt{a}]\!], \underline{0}) \mathbin{\hat{+}} f$$

$$\mathtt{qet}\big[\mathtt{x = e}\big]\{f\} \triangleq f[\mathtt{x} := \mathtt{e}] \qquad \mathtt{qet}\big[\mathtt{stm}_1; \mathtt{stm}_2\big]\{f\} \triangleq \mathtt{qet}\big[\mathtt{stm}_1\big]\{\mathtt{qet}\big[\mathtt{stm}_2\big]\{f\}\}$$

$$\mathtt{qet}\big[\overline{\mathtt{q}} \mathrel{*}= \mathtt{U}\big]\{f\} \triangleq f[\mathtt{U}_{\overline{\mathtt{q}}}] \qquad \mathtt{qet}\big[\mathtt{if(b)\{stm}_1\mathtt{\} else \{stm}_2\mathtt{\}}\big]\{f\} \triangleq \mathtt{qet}\big[\mathtt{stm}_1\big]\{f\} +_{[\![\mathtt{b}]\!]} \mathtt{qet}\big[\mathtt{stm}_2\big]\{f\}$$

$$\mathtt{qet}\big[\mathtt{while(b)\{stm\}}\big]\{f\} \triangleq \mathsf{lfp}\left(\lambda F.\mathtt{qet}\big[\mathtt{stm}\big]\{F\} +_{[\![\mathtt{b}]\!]} f\right)$$

Figure 4. Quantum Expectation Transformer $\mathtt{qet}\big[\cdot\big]\{\cdot\} : \mathtt{Program} \to \mathsf{S}^{\mathtt{State}} \to \mathsf{S}^{\mathtt{State}}$.

continuity	$\mathtt{qet}\big[\mathtt{stm}\big]\{\sup_i f_i\} = \sup_i \mathtt{qet}\big[\mathtt{stm}\big]\{f_i\}$ for any ω-chain $(f_i)_i$
monotonicity	$f \leq g \implies \mathtt{qet}\big[\mathtt{stm}\big]\{f\} \leq \mathtt{qet}\big[\mathtt{stm}\big]\{g\}$
linearity	$p \in [0,1] \implies \mathtt{qet}\big[\mathtt{stm}\big]\{f +_p g\} = \mathtt{qet}\big[\mathtt{stm}\big]\{f\} +_p \mathtt{qet}\big[\mathtt{stm}\big]\{g\}$
upper invariant	$([\![\neg \mathtt{b}]\!] \cdot f \leq g \ \wedge\ [\![\mathtt{b}]\!] \cdot \mathtt{qet}\big[\mathtt{stm}\big]\{g\} \leq g) \implies \mathtt{qet}\big[\mathtt{while(b)\{stm\}}\big]\{f\} \leq g$

Figure 5. Universal laws derivable for the quantum expectation transformer.

It holds that $\mathtt{H}\binom{\alpha}{\beta} = \frac{\alpha+\beta}{\sqrt{2}}|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}|1\rangle$ and that:

$$\begin{aligned}
&\mathtt{qect}\big[\mathtt{stm}_0\big]\{g\}\,(s, \tbinom{\alpha}{\beta}) \\
&\quad = g_3(s, \tbinom{\alpha}{\beta}) \\
&\quad = \frac{|\alpha+\beta|^2}{2}(1+0) + \frac{|\alpha-\beta|^2}{2}(1+1+(0-1)^2) \\
&\quad = 1 + |\alpha-\beta|^2 = g(s, \tbinom{\alpha}{\beta}).
\end{aligned}$$

Therefore, (in)equalities (8) and (9) are satisfied by g. It follows that

$$\begin{aligned}
\mathtt{qect}\big[CT(\mathtt{q})\big]\{\underline{0}\} &\leq \mathtt{qect}\big[\mathtt{x}^{\mathcal{B}} = \mathtt{true}\big]\{g\} \\
&= g[\mathtt{x} := 1] = \lambda(s, \tbinom{\alpha}{\beta}).1 + |\alpha-\beta|^2
\end{aligned}$$

is the cost expectation of the program. Note that, in this case, this bound is exact.

4.3 Soundness and Adequacy

We first give a semantic counterpart to the quantum expectation transformer. To this end, for $f \in \mathsf{S}^{\mathtt{State}}$ let $\mathtt{QET}\{f\} \in \mathsf{S}^{\mathtt{Conf} \cup \mathtt{State}}$ be the least function (in the pointwise order inherited from S), such that:

$$\mathtt{QET}\{f\}(\mu) = \begin{cases} f(\mu) & \text{if } \mu \in \mathtt{State} \\ c \mathbin{\hat{+}} \mathbb{E}_\delta(\mathtt{QET}\{f\}) & \text{if } \mu \in \mathtt{Conf} \text{ and } \mu \xrightarrow{c} \delta. \end{cases}$$

Finally, we overload notation and set

$$\mathtt{QET}[\mathtt{stm}]\{f\}(\sigma) \triangleq \mathtt{QET}\{f\}(\mathtt{stm}, \sigma).$$

The following correspondence is not difficult to establish.

Lemma 4.5. *For all* $\mathtt{stm} \in \mathtt{Statement}$ *and* $f \in \mathsf{S}^{\mathtt{State}}$, $\mathtt{QET}[\mathtt{stm}]\{f\} = \mathsf{ecost}_{\mathtt{stm}} \mathbin{\hat{+}} \mathsf{evalue}_{\mathtt{stm}}(f)$.

We now show that the quantum expectation transformer coincides with its semantic counterpart. Via this correspondence, the above lemma allows us to relate the quantum expectation transformer — and its derivates from Definition 4.2 — to the cost and the semantics of the considered programs. To establish the link, we make use of the following two identities.

Lemma 4.6. *The following identities hold.*

1. $\mathtt{QET}[\mathtt{stm}_1; \mathtt{stm}_2]\{f\} = \mathtt{QET}[\mathtt{stm}_1]\{\mathtt{QET}[\mathtt{stm}_2]\{f\}\}$*; and*
2. $\mathtt{QET}[\mathtt{while(b)\{stm\}}]\{f\} = \mathsf{lfp}(\lambda F.\mathtt{QET}[\mathtt{stm}]\{F\} +_{[\![\mathtt{b}]\!]} f)$.

Theorem 4.7 (Soundness)**.** *For all* $\mathtt{stm} \in \mathtt{Statement}$, $\sigma \in \mathtt{State}$ *and* $f \in \mathsf{S}^{\mathtt{State}}$, $\mathtt{qet}\big[\mathtt{stm}\big]\{f\}\,(\sigma) = \mathtt{QET}[\mathtt{stm}]\{f\}(\sigma)$.

Proof. The theorem is proven by induction on $\mathtt{stm}$. Almost all cases follow by definition. The only two non-trivial cases, those of command composition and loops, follow from the induction hypothesis by using Lemma 4.6. □

This theorem and Lemma 4.5 immediately show how to recover the expected cost and expected value of programs.

Corollary 4.8 (Adequacy)**.** *The following identities hold, for all* $\mathtt{stm} \in \mathtt{Statement}$, $\sigma \in \mathtt{State}$ *and* $f \in \mathsf{S}^{\mathtt{State}}$.

1. $\mathtt{qect}\big[\mathtt{stm}\big]\{\underline{0}\}\,(\sigma) = \mathsf{ecost}_{\mathtt{stm}}(\sigma)$*; and*
2. $\mathtt{qev}_{\mathsf{S}}\big[\mathtt{stm}\big]\{f\}\,(\sigma) = \mathsf{evalue}_{\mathtt{stm}}(f)(\sigma)$.

Example 4.9. We illustrate the soundness theorem on our simple leading example (Example 3.3). As calculated in Example 3.3 and Example 4.4, we have

$$\mathsf{ecost}_{CT(\mathsf{q})} = \lambda(s, \begin{pmatrix}\alpha\\ \beta\end{pmatrix}).1 + |\alpha - \beta|^2 = \mathsf{qect}\left[CT(\mathsf{q})\right]\left\{\underline{0}\right\}.$$

4.4 Relationship to Denotational Semantics

As a special case of our quantum expectation transformer, we can define a quantum denotational semantics for our language. Recall that the formation conditions for configurations are defined with respect to three sets B, V, Q of Boolean, numerical and quantum variables, respectively. Let $B = \{b_1, \ldots, b_n\}$, $V = \{v_1, \ldots v_m\}$ and $Q = \{q_1, \ldots q_k\}$. We can define a Kegelspitze K which serves as the semantic domain for our programs by setting $\mathsf{K} \triangleq \{0,1\}^n \times \mathbb{Z}^m \to D_{2^k}$, where the order and convex structure of K is inherited pointwise from D_{2^k} (see Example 2.6). A cost structure $\mathcal{K}$ is now obtained by equipping K with the forgetful cost addition $\mathcal{K} = (\mathsf{K}, +_{\mathsf{f}})$. With this choice of cost structure, our quantum expectation transformer qet from Definition 4.1 yields a *quantum denotational semantics* transformer

$$\mathsf{qev}_{\mathsf{K}}\left[\cdot\right]\left\{\cdot\right\} : \mathsf{Program} \to \mathsf{K}^{\mathsf{State}} \to \mathsf{K}^{\mathsf{State}}.$$

Recall that a quantum denotational semantics consists in giving a mathematical interpretation of program configurations which is invariant under the operational semantics (in a probabilistic sense). This can be obtained from $\mathsf{qev}_{\mathsf{K}}$ by making a suitable choice for the continuation. In particular, if we choose

$$h : \mathsf{State} \to \mathsf{K}$$
$$h(s^B, s^V, |\varphi\rangle) = \lambda((t_1, \ldots, t_n), (u_1, \ldots, u_m)).$$
$$\begin{cases} |\varphi\rangle\langle\varphi| & \text{if } t_i = s^B(b_i) \text{ for } 1 \le i \le n \text{ and} \\ & u_j = s^V(v_j) \text{ for } 1 \le j \le m \\ \mathbf{0} & \text{otherwise} \end{cases}$$

then, a quantum denotational semantics

$$(\!|-|\!) : \mathsf{Conf} \cup \mathsf{State} \to \mathsf{K}$$

can be defined by $(\!|(\mathsf{stm}, \sigma)|\!) \triangleq \mathsf{qev}_{\mathsf{K}}\left[\mathsf{stm}\right]\left\{h\right\}(\sigma)$ for configurations and $(\!|\sigma|\!) \triangleq h(\sigma)$ for program states (which are our notion of terminal objects). Then, by Corollary 4.8, for any well-formed configuration $\mu = (\mathsf{stm}, \sigma)$ we have that

$$(\!|\mu|\!) = \mathsf{qev}_{\mathsf{K}}\left[\mathsf{stm}\right]\left\{h\right\}(\sigma) = \mathsf{evalue}_{\mathsf{stm}}(h)(\sigma) = \mathbb{E}_{\mathsf{nf}_{\to}(\mu)}((\!|-|\!)).$$

This shows that the denotational interpretation $(\!|\mu|\!)$ is equal to the (countable) convex sum of the interpretations of final states (i.e., terminal objects) that μ can reduce to. In this equation, each probability weight associated to a final state τ is given by the reduction probability of μ to τ as determined by the operational semantics. This is precisely the statement of *strong adequacy* in the denotational semantics of probabilistic [??] and quantum programming languages [??].

```
RUS(q') ≜ x^𝓑 = true;
        while(x){
          q = |+⟩;                  ⎫            ⎫
          q *= T;                   ⎪            ⎪
          consume(1);               ⎪            ⎪
          q, q' *= CNOT;            ⎪            ⎪
          q *= H;                   ⎬ stm_0      ⎬ stm
          q, q' *= CNOT;            ⎪            ⎪
          q *= T;                   ⎪            ⎪
          consume(1);               ⎪            ⎪
          q *= H;                   ⎭            ⎪
          x = meas(q)                            ⎭
        }
```

Listing 2. Repeat until success.

5 Illustrating Examples

In this section we present more intricate examples illustrating how cost analysis can be performed for quantum algorithms. The analysis has a focus on expected costs. Hence, the cost structure is fixed to be $(\mathbb{R}^{+\infty}, +)$. Consequently, cost expectations will be functions in the set $(\mathbb{R}^{+\infty})^{\mathsf{State}}$.

Notations. Throughout the following, we denote by κ a *classical expectation*, i.e., an expectation which satisfies $\kappa(s, |\varphi\rangle) = \kappa(s, |\psi\rangle)$ for all $s, |\varphi\rangle$, and $|\psi\rangle$. A classical expectation κ thus only depends on the classical state.

We also define the following syntactic sugar:

- $\mathsf{q} = |0\rangle$ for $\mathsf{x}^{\mathcal{B}} = \mathsf{meas(q)}$; $\mathsf{if(x)\{\,q \mathrel{*}= X\}\ else\ \{skip\}}$, where X is the unitary operator for negation (Pauli-X gate), defined by $\mathsf{X}\,|0\rangle = |1\rangle$ and $\mathsf{X}\,|1\rangle = |0\rangle$.
- $\mathsf{q} = |+\rangle$ for $\mathsf{q} = |0\rangle$; $\mathsf{q} \mathrel{*}= \mathsf{H}$.

5.1 Repeat until success

In this example, we consider a program that implements a Repeat-until-success algorithm and we show that our analysis can be used to infer upper bounds on the expected *T-count*, i.e., the expected number of times the so-called T gate is used.

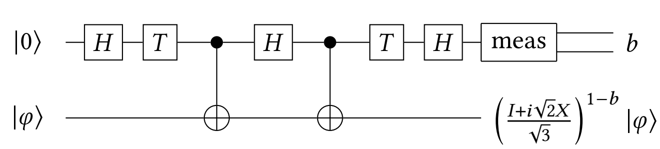

Figure 6. A quantum circuit illustrating a repeat-unitl-success pattern.

Repeat-until-success [?] can be used to implement quantum unitary operators by using repeated measurements. An advantage of this approach is that this often allows us to implement quantum unitary operators by using fewer T gates, which are costly to implement fault-tolerantly [??].

Consider the example in Listing 2. This quantum algorithm will repeatedly execute the quantum operations described by the quantum circuit in Figure 6, as specified in [?, Figure 8], where the operators T and CNOT correspond to the following quantum gates:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \quad CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

After measuring the first (ancilla) qubit q, there are two possibilities. With probability $1/4$, we measure 1 and then the state of the second qubit q' is again $|\varphi\rangle$ and we repeat the algorithm. With probability $3/4$, we measure 0 and then the algorithm terminates and the second qubit is now in state $\frac{I+i\sqrt{2}X}{\sqrt{3}}|\varphi\rangle$.

Analysis. Let $\mathtt{stm} \triangleq \mathtt{stm}_0; \mathtt{x} = \mathtt{meas}(\mathsf{q})$ be the body of the while loop statement in the above program. For any classical (cost) expectation $\kappa \in (\mathbb{R}^{+\infty})^{\mathtt{State}}$, by rules of Figure 4, it holds that

$$\mathtt{qect}\big[\mathtt{stm}_0\big]\{\kappa\} = \kappa + \underline{2}.$$

Indeed, one can check easily that each qubit operation in $\mathtt{stm}_0$ leaves the expectation κ unchanged. For example, we have that $\mathtt{qect}\big[\mathsf{q} \mathrel{*}= \mathtt{H}\big]\{\kappa\} = \kappa[\mathtt{H}] = \kappa$. Therefore, we just have to take into account the two $\mathtt{consume}(1)$ statements.

As the probability of measuring 0 is constant for each iteration ($\frac{3}{4}$, see [?]), for any cost expectation κ, it holds that:

$$\begin{aligned}\mathtt{qect}\big[\mathtt{x} = \mathtt{meas}(\mathsf{q})\big]\{\kappa\} &= \kappa[\mathtt{x} := 0; \mathtt{M}_0^{\mathsf{q}}] +_{\frac{3}{4}} \kappa[\mathtt{x} := 1; \mathtt{M}_1^{\mathsf{q}}] \\ &= \underline{3/4} \cdot \kappa[\mathtt{x} := 0] + \underline{1/4} \cdot \kappa[\mathtt{x} := 1]\end{aligned}$$

Putting this all together, the expectation of stm is:

$$\begin{aligned}\mathtt{qect}\big[\mathtt{stm}\big]\{\kappa\} &= \mathtt{qect}\big[\mathtt{stm}_0\big]\big\{\mathtt{qect}\big[\mathtt{x} = \mathtt{meas}(\mathsf{q})\big]\{\kappa\}\big\} \\ &= \mathtt{qect}\big[\mathtt{x} = \mathtt{meas}(\mathsf{q})\big]\{\kappa\} + \underline{2} \\ &= \underline{3/4} \cdot \kappa[\mathtt{x} := 0] + \underline{1/4} \cdot \kappa[\mathtt{x} := 1] + \underline{2}.\end{aligned}$$

By Law (*upper invariant*) of Figure 5, it suffices to find an expectation κ satisfying the following inequalities

$$[\![\neg \mathtt{x}]\!] \cdot \underline{0} \le \kappa \tag{10}$$

$$[\![\mathtt{x}]\!] \cdot \mathtt{qect}\big[\mathtt{stm}\big]\{\kappa\} \le \kappa \tag{11}$$

in order to compute (an upper bound on) the cost expectation $\mathtt{qect}\big[\mathtt{while}(\mathtt{x})\{\mathtt{stm}\}\big]\{\underline{0}\}$. Inequalities (10) and (11) are satisfied by setting $\kappa \triangleq [\![\mathtt{x}]\!] \cdot \underline{\frac{8}{3}}$.

We conclude by computing the expectation of the whole program

$$\begin{aligned}\mathtt{qect}\big[RUS(\mathsf{q}')\big]\{\underline{0}\} &\le \mathtt{qect}\big[\mathtt{x}^{\mathcal{B}} = \mathtt{true}\big]\{\kappa\} \\ &= \kappa[\mathtt{x} := 1] = \underline{8/3}.\end{aligned}$$

The expected cost (the expected number of T gates used) of this algorithm is bounded by $\frac{8}{3}$. Note that this bound is tight.

5.2 Chain of k entangled qubits

The following example illustrates that the presented cost analysis can also deal with nested while loops on a non-trivial example using classical data. We consider a simple algorithm attempting to prepare a large entangled state, namely a graph state represented by a path on k qubits, i.e. $|\phi_k\rangle = \prod_{i=0}^{k-2} CZ_{i,i+1} \otimes_{j=0}^{k-1} |+\rangle$ where CZ is the following 2-qubit unitary transformation:

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

One can prepare the desired state by initializing k qubits in a row in the state $|+\rangle$ and then applying the CZ gate $k-1$ times, one for each pair of consecutive qubits. Notice that the order in which the CZ are applied is irrelevant (as they are commuting). However, in some settings, like linear optical quantum computing, CZ cannot be implemented deterministically. Nielsen [?] showed that CZ can be implemented in linear optics with probability of success $1/4$. The case of a failure corresponds to a measurement of the corresponding two qubits.

Notations. In this example, we use $F(\overline{\mathsf{q}}, \overline{\mathtt{x}})$ as a shorthand notation for a (non-recursive) call to program F with parameters $\overline{\mathsf{q}}, \overline{\mathtt{x}}$. A call to $F(\overline{\mathsf{q}}, \overline{\mathtt{x}})$ consists in unfolding the statement of F after a careful variable renaming, avoiding name clashes. We also assume that, for a given sequence of qubits $\mathsf{q}_1, \ldots, \mathsf{q}_n$, we can access the i-th qubit through a call $\mathsf{q}_{\mathtt{x}}$, provided that variable x holds the value i in the store.

The program $FUSE(\mathsf{q}, \mathsf{q}', \mathtt{x})$ in Listing 3 models the entanglement of two input qubits q and q' in state $|+\rangle$ with probability $1/4$. The Boolean variable x records whether this operation succeeded. The $CHAIN4(\mathsf{q}_1, \mathsf{q}_2, \mathsf{q}_3, \mathsf{q}_4)$ from Listing 5 entangles four given qubits, by iterating $FUSE$ until eventually all links have been established sucessfully. The general algorithm in Listing 5 then makes use of this procedure by iteratively appending 4-entangled-qubits chains to the main chain, resulting eventually in a chain of $k \le t \le k+3$ entangled qubits.

Analysis. Let us first consider the sub-program $FUSE$ from Listing 3. Using laws of Figure 4, it is not difficult to see, that for any classical expectation κ,

$$\begin{aligned}&\mathtt{qect}\big[FUSE(\mathsf{q}, \mathsf{q}', \mathtt{x})\big]\{\kappa\} = \\ &\qquad \underline{1} + \underline{1/4} \cdot \kappa[\mathtt{x} := \mathtt{true}] + \underline{3/4} \cdot \kappa[\mathtt{x} := \mathtt{false}].\end{aligned}$$

This can be formally verified by unfolding definitions, exploiting that κ does not depend on the quantum state. Let us turn our attention to $CHAIN4$ from Listing 4, and observe

$$\mathtt{qect}\big[\mathtt{stm}_2\big]\{\kappa\} \le \underline{4} + \kappa,$$

```
FUSE(q, q′, x) ≜
  consume(1);
  a^Q = |+⟩;
  b^Q = |+⟩;
  x = meas(a);
  y = meas(b);
  if(x ∧ y){ // with probability 1/4
    q, q′ *= CZ;
    x = true // Flag set to success
  } else { // with probability 3/4
    x = meas(q);
    x = meas(q′);
    x = false // Flag set to failure
  }
```

Listing 3. Applying a CZ gate to q, q′ with probability 1/4.

```
CHAIN4(q1, q2, q3, q4) ≜
  x^B = false;
  while(¬x){
    x1^B = false;  // Left pair
    while(¬x1){
      q1 = |+⟩;
      q2 = |+⟩;
      FUSE(q1, q2, x1)
    };                                   } stm1
    x2^B = false;  // Right pair
    while(¬x2){
      q3 = |+⟩;
      q4 = |+⟩;
      FUSE(q3, q4, x2)
    };                                   } stm2
    FUSE(q2, q3, x) // Fusion of pairs
  }
```

Listing 4. Chaining four qubits.

```
CHAIN(k, q0, ..., q_{k+3}) ≜
  t^V = 0;
  q0 = |+⟩;
  x^B = false;
  while(0 ≤ t ∧ t < k){
    CHAIN4(q_{t+1}, q_{t+2}, q_{t+3}, q_{t+4});
    FUSE(q_t, q_{t+1}, x);
    if(x){t = t + 4} else {t = t − 1};   } stm
    if(t = −1){t = 0; q0 = |+⟩} else {skip}
  }
```

Listing 5. Create a chain of k entangled qubits.

when κ is independent of x_2 assigned in $\mathtt{stm}_2$, i.e., $\kappa[x_2 := b] = \kappa$. To see this, take $g \triangleq [\![\neg x_2]\!] \cdot \underline{4} + \kappa$, and hence

$$\begin{aligned}
&\mathtt{qect}\left[\begin{matrix} q_3 = |+\rangle\,;\, q_4 = |+\rangle\,; \\ FUSE(q_3, q_4, x_1)\end{matrix}\right]\{g\} \\
&\quad = \mathtt{qect}\left[q_3 = |+\rangle\,;\, q_4 = |+\rangle\right]\left\{\begin{matrix}\underline{1} + \underline{1/4} \cdot g[x_2 := \mathtt{true}] \\ + \underline{3/4} \cdot g[x_2 := \mathtt{false}]\end{matrix}\right\} \\
&\quad = \underline{4} + \kappa.
\end{aligned}$$

Now it is clear that, since $[\![\neg x_2]\!] \cdot \underline{4} + \kappa \leq [\![\neg x_2]\!] \cdot \underline{4} + \kappa$ holds trivially, g constitutes an upper-invariant of the while loop in $\mathtt{stm}_2$, see Law (*upper invariant*). Substituting false for x_2 gives the bound for $\mathtt{stm}_2$. The same argument shows that

$$\mathtt{qect}\left[\mathtt{stm}_1\right]\{\kappa\} \leq \underline{4} + \kappa,$$

for any κ independent of x_1.

Concerning the outer loop, let κ now refer to a classical cost function independent of the Boolean variables x_1 and x_2. Putting things together,

$$\begin{aligned}
\mathtt{qect}\left[\mathtt{stm}_1; \mathtt{stm}_2; FUSE(q_2, q_3, x)\right]\{[\![\neg x]\!] \cdot \underline{36} + \kappa\} \\
\leq \underline{4} + \underline{4} + \underline{28} + \kappa = \underline{36} + \kappa,
\end{aligned}$$

and finally

$$\mathtt{qect}\left[CHAIN4(q_1, q_2, q_3, q_4)\right]\{\kappa\} \leq \underline{36} + \kappa,$$

via Law (*upper invariant*). Concerning the overall code from Listing 5, let us now define the classical expectation

$$f \triangleq [\![0 \leq t \wedge t < k + 4]\!] \cdot \underline{148} \cdot (\underline{k} - [\![t]\!] + \underline{4}).$$

We obtain

$$\begin{aligned}
&\mathtt{qect}\left[\mathtt{stm}\right]\{f\} \\
&= \underline{36} + \underline{1} + \underline{1/4} \cdot [\![t \neq -1]\!] \cdot f[t := t + 4] \\
&\quad + \underline{3/4} \cdot ([\![t = -1]\!] \cdot f[t := 0] + [\![t \neq -1]\!] \cdot f[t := t - 1]) \\
&= \underline{37} + [\![0 \leq t \wedge t < k + 4]\!] \cdot (\underline{1/4} \cdot (\underline{148} \cdot (\underline{k} - [\![t]\!])) \\
&\quad + \underline{3/4} \cdot \underline{148} \cdot (\underline{k} - [\![t]\!] + \underline{5})).
\end{aligned}$$

Exploiting the loop-guard $0 \leq t < k$, we finally establish that f is an upper-bound to the expected runtime of the loop, where the required inequality is in particular encompassed by the inequality

$$\begin{aligned}
[\![0 \leq t \wedge t < k]\!] \cdot \underline{37} + \underline{37} \cdot (\underline{k} - [\![t]\!]) + \underline{111} \cdot (\underline{k} - [\![t]\!] + \underline{5}) \\
\leq \underline{148} \cdot (\underline{k} - [\![t]\!] + \underline{4}),
\end{aligned}$$

which can be easily seen to hold. Substituting $\underline{0}$ for $[\![t]\!]$ in this expectation, we conclude that the overall expected cost is bounded by $148 \times (k + 4)$.

5.3 Quantum walk

In this last example, we consider the Hadamard quantum walk on an n-circle as defined in [?, Section VI.B]. Our goal is to illustrate on a non-trivial example that the cost analysis may depend directly on the program quantum state, as in Example 3.1.

Let q be a quantum bit of the 2-dimensional state space $\mathcal{H}_q$, whose basis states $|L\rangle$ and $|R\rangle$ indicate directions Left and Right, respectively. Let $\mathcal{H}_p$ be an n-dimensional Hilbert space with orthonormal basis $|0\rangle, |1\rangle, \ldots, |n-1\rangle$ for the positions. The state space $\mathcal{H}$ for the quantum walk is defined by $\mathcal{H} \triangleq \mathcal{H}_q \otimes \mathcal{H}_p$. The program itself is given in Listing 6. The operator S shifts the position depending of the direction

```
x^B = true;
while(x){
  x = meas(p);
  q *= H;
  q,p *= S;
  consume(1)
}
```

Listing 6. Quantum walk.

state and is defined by the following standard unitary operator: $S = \Sigma_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \ominus 1\rangle\langle i| + \Sigma_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|$, where $\oplus$ and $\ominus$ denote addition and substraction modulo n.

Adaptation. Up to now, for simplicity, we only considered quantum systems composed of qubits and measurement in the computational basis. However, for this example, it is more convenient to consider more general systems and measurements. Because of this, we adapt slightly the operational semantics and expected cost transformer to this particular setting. Any quantum state $|\varphi\rangle$ can be written as

$$|\varphi\rangle \triangleq \sum_{i=0}^{n-1} a_i |L\rangle|i\rangle + \sum_{i=n}^{2n-1} a_i |R\rangle|i-n\rangle = \begin{pmatrix} a_0 \\ \vdots \\ a_{2n-1} \end{pmatrix},$$

for $n \geq 1$ and for $a_i \in \mathbb{C}$ such that $\Sigma_i |a_i|^2 = 1$. The probability that the quantum state $|\varphi\rangle$ is at position 0 is given by $p_0^{\mathsf{p}}|\varphi\rangle \triangleq \langle\varphi|(I_2 \otimes |0\rangle\langle 0|)|\varphi\rangle$. The probability that the quantum state $|\varphi\rangle$ is at a position distinct from 0 is given by $p_{\neq 0}^{\mathsf{p}}|\varphi\rangle \triangleq \langle\varphi|(I_{2n} - I_2 \otimes |0\rangle\langle 0|)|\varphi\rangle$. These two probabilities trivially satisfy $p_0^{\mathsf{p}} + p_{\neq 0}^{\mathsf{p}} = \underline{1}$ and it holds that $p_{\neq 0}|\varphi\rangle = 1 - (|a_0|^2 + |a_n|^2) = \sum_{i\neq 0, i\neq n} |a_i|^2$.

We adapt in a direct and obvious way the result of the calculation of a measurement to the n-dimensional case: the result of measuring the outcome 0 (`false`), $\mathsf{M}_0^{\mathsf{p}}$, and the result of measuring an outcome distinct from 0 (`true`), $\mathsf{M}_{\neq 0}^{\mathsf{p}}$, are defined by $\mathsf{M}_0^{\mathsf{p}} \triangleq \frac{1}{\sqrt{p_0^{\mathsf{p}}}} I_2 \otimes |0\rangle\langle 0|$ and $\mathsf{M}_{\neq 0}^{\mathsf{p}} \triangleq \frac{1}{\sqrt{p_{\neq 0}^{\mathsf{p}}}}(I_{2n} - I_2 \otimes |0\rangle\langle 0|)$, respectively.

The operational semantics of Figure 3 and the expected cost transformer of Figure 4 can be adapted straightforwardly to this new setting. E.g., the rule of Figure 4 for measurement on qubit p is rewritten as follows:

$$\mathtt{qet}\big[\mathtt{x} = \mathtt{meas(p)}\big]\{f\} = f[\mathtt{x} := 0; \mathsf{M}_0^{\mathsf{q}}] +_{p_0^{\mathsf{q}}} f[\mathtt{x} := 1; \mathsf{M}_{\neq 0}^{\mathsf{q}}].$$

All the other rules remain unchanged. Our soundness results still hold in this context. In particular, Theorem 4.3 and Corollary 4.8 are still valid.

Analysis. In order to analyse the expected cost of the above program, we search for an expectation g satisfying the prerequisite for applying Law (*upper invariant*):

$$[\![\mathtt{x}]\!] \cdot \mathtt{qect}\begin{bmatrix} \mathtt{x = meas(p);} \\ \mathtt{q *= H;} \\ \mathtt{q,p *= S;} \\ \mathtt{consume(1)} \end{bmatrix}\{g\} \leq g. \tag{12}$$

Using the (adapted) laws of Figure 4, the following equalities can be derived:

$$\begin{aligned}
\mathtt{qect}\big[\mathtt{consume(1)}\big]\{g\} &= \underline{1} + g \triangleq g_1 \\
\mathtt{qect}\big[\mathtt{q,p *= S_{q,p}}\big]\{g_1\} &= (\underline{1} + g)[\mathsf{S}_{\mathtt{q,p}}] \triangleq g_2 \\
\mathtt{qect}\big[\mathtt{q *= H_q}\big]\{g_2\} &= g_2[\mathsf{H}_{\mathtt{q}}] = (\underline{1} + g)[\mathsf{S}_{\mathtt{q,p}}\mathsf{H}_{\mathtt{q}}] \triangleq g_3 \\
\mathtt{qect}\big[\mathtt{x = meas(p)}\big]\{g_3\} &= g_3[\mathtt{x} := 0; \mathsf{M}_0^{\mathsf{p}}] +_{p_0^{\mathsf{p}}} g_3[\mathtt{x} := 1; \mathsf{M}_{\neq 0}^{\mathsf{p}}]
\end{aligned}$$

where the matrices corresponding to the operators $\mathsf{S}_{\mathtt{q,p}}$ and $\mathsf{H}_{\mathtt{q}}$ are equal to $H \otimes I_n$ and S, respectively.

In a nutshell, $\mathtt{qect}\big[\mathtt{x = meas(p)}\big]\{g_3\}$ can be written as:

$$\begin{aligned}
\mathtt{qect}\big[\mathtt{x = meas(p)}\big]\{g_3\} = p_0^{\mathsf{p}} \cdot (\underline{1} + g)[\mathsf{S}_{\mathtt{q,p}}\mathsf{H}_{\mathtt{q}}][\mathtt{x} := 0; \mathsf{M}_0^{\mathsf{p}}] \\
+ p_{\neq 0}^{\mathsf{p}} \cdot (\underline{1} + g)[\mathsf{S}_{\mathtt{q,p}}\mathsf{H}_{\mathtt{q}}][\mathtt{x} := 1; \mathsf{M}_{\neq 0}^{\mathsf{p}}]
\end{aligned}$$

and Equation (12) can be simplified as follows:

$$p_{\neq 0}^{\mathsf{p}}(\underline{1} + g)[\mathtt{x} := 1; \mathsf{S}_{\mathtt{q,p}}\mathsf{H}_{\mathtt{q}}\mathsf{M}_{\neq 0}^{\mathsf{p}}] \leq g. \tag{13}$$

Equation (13) holds if for any store s and any quantum state $|\varphi\rangle$, we have:

$$p_{\neq 0}^{\mathsf{p}}|\varphi\rangle\,(1 + g(s[\mathtt{x} := 1], \mathsf{S}_{\mathtt{q,p}}\mathsf{H}_{\mathtt{q}}\mathsf{M}_{\neq 0}^{\mathsf{p}}|\varphi\rangle)) \leq g(s, |\varphi\rangle).$$

Notice that $|\psi\rangle \triangleq \mathsf{S}_{\mathtt{q,p}}\mathsf{H}_{\mathtt{q}}\mathsf{M}_{\neq 0}^{\mathsf{p}}|\varphi\rangle$ will be the quantum state entering the loop in the next iteration.

For any complex number $z \in \mathbb{C}$, z^* will denote the complex conjugate of z and the real part of z will be denoted by $\mathfrak{R}(z)$.

Consider the expectation g_n defined below (and inspired by [**?**]).

$$g_n(s, \begin{pmatrix} a_0 \\ \vdots \\ a_{2n-1} \end{pmatrix}) \triangleq \sum_{i=0}^{2n-1} f_n(i)|a_i|^2 + 2\sum_{j=0}^{2n-1}\sum_{k=0}^{2n-1} h_n(j,k)\mathfrak{R}(a_j a_k^*)$$

with $f_n(i) \triangleq \begin{cases} i(n-i)+1 & \text{if } 0 \leq i \leq n-1 \\ (i-n)(2n-i)+1 & \text{if } n \leq i \leq 2n-1 \end{cases}$

and with $h_n(j,k) \triangleq$

$$\begin{cases}
(-1)^{\frac{j-k}{2}} k(n-1-j) & \text{if } \begin{cases} k,j \in [0, n-1], \\ k < j, \\ (j-k)\ \%\ 2 = 0 \end{cases} \\
(-1)^{\frac{j-k}{2}} (2n-j)(k-n-1) & \text{if } \begin{cases} k,j \in [n, 2n-1], \\ k < j, \\ (j-k)\ \%\ 2 = 0 \end{cases} \\
(-1)^{\frac{k-j-n}{2}} (j+k-2n) & \text{if } \begin{cases} j+n, k \in [n+1, 2n-1], \\ j+n \leq k, \\ (k-(j+n))\ \%\ 2 = 0 \end{cases} \\
0 & \text{otherwise.}
\end{cases}$$

g_n is a solution of the inequality in Equation (13). This can be shown by symbolically computing the substraction of the left-hand side and the right-hand side of the inequality.

Now we consider the simple case where $n = 2$. For s such that $[\![\mathrm{x}]\!]^s = 1$, Equation (13) can be rewritten as:

$$(|a_1|^2+|a_3|^2)(1+g(s, \frac{1}{\sqrt{2(|a_1|^2+|a_3|^2)}}\begin{pmatrix} a_1+a_3 \\ 0 \\ a_1-a_3 \\ 0 \end{pmatrix})) \leq g(s, \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}).$$

The above inequality is satisfied for $g_2(s, |\varphi\rangle) \triangleq 1+|a_1|^2+|a_3|^2$. Hence, starting in position $|1\rangle$ (i.e., $a_1 |L\rangle |1\rangle + a_3 |R\rangle |1\rangle$, with $|a_1|^2+|a_3|^2 = 1$), the expected cost is 2, whereas starting in position $|0\rangle$, the expected cost is 1 (as $a_1 = a_3 = 0$).

In the case, where $n = 3$, the expectation $g_3(s, |\varphi\rangle) \triangleq 2 - |a_0|^2-|a_3|^2+|a_2+a_5|^2+|a_1-a_4|^2$ is a solution to Equation (13).

Note that our expectations g_n can be recovered from the matrices Q_n in the work of [?, Section VI.B] as follows: $g_n(s, |\varphi\rangle) \triangleq \langle\varphi| Q_n |\varphi\rangle$.

6 Conclusion and Future Work

We presented an adequate notion of quantum expectation transformer and showed through practical examples that it can be used to find the expected cost of quantum programs and to approximate it by inferring upper bounds as well.

Concerning the calculus's power and limitations, the problem of finding the expectation of a given program is indeed undecidable as it provides an adequate denotational semantics. This is of course unavoidable because the problem of finding the expected cost of a program is also undecidable. Quantum expectation transformers are also clearly more intricate than classical and probabilistic transformers because of the consideration of the quantum state. In general, the probability depends on the quantum state which increases the computational difficulty of computing the expectation. While this problem is clearly undecidable in general, a restriction to a well-defined function space for expectations may allow for an effective solution, at the price of incompleteness. Existing work in the literature on automation of expected cost transfomers or related work for *classical* programs cf. [????] should provide ample guidance in this respect. Because of this, we believe that our qet-calculus provides a principled foundation for automation.

Acknowledgments

This work is supported by the Inria associate team TC(Pro)^3, by the HORIZON 2020 project NEASQC (NExt ApplicationS of Quantum Computing), by the STIC-AmSud project QAPLA: "Quantum aspects of programming languages", and by the ANR Project PPS: "Probabilistic Program Semantics".

References

Martin Avanzini, Gilles Barthe, and Ugo Dal Lago. 2021. On continuation-passing transformations and expected cost analysis. *Proc. of the ACM on Programming Languages* 5, ICFP (2021), 1–30. https://doi.org/10.1145/3473592

Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. 2020a. On probabilistic term rewriting. *Science of Computer Programming* 185 (2020). https://doi.org/10.1016/j.scico.2019.102338

Martin Avanzini, Georg Moser, and Michael Schaper. 2020b. A modular cost analysis for probabilistic programs. *Proc. of the ACM on Programming Languages* 4, OOPSLA (2020), 172:1–172:30. https://doi.org/10.1145/3428240

Olivier Bournez and Florent Garnier. 2005. Proving Positive Almost-Sure Termination. In *Proc. of RTA 2005 (LNCS, Vol. 3467)*. Springer, 323–337. https://doi.org/10.1142/S0129054112400588

Sergey Bravyi and Alexei Kitaev. 2005. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A* 71, 2 (2005), 022316. https://doi.org/10.1103/PhysRevA.71.022316

Edsger W. Dijkstra. 1976. *A discipline of programming*. Prentice-Hall Englewood Cliffs.

David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. 2014. An algorithm for the T-count. *Quantum Information & Computation* 14, 15-16 (2014), 1261–1276. https://doi.org/10.26421/QIC14.15-16-1

Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. 2014. Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Performance Evaluation* 73 (2014), 110–132. https://doi.org/10.1016/j.peva.2013.11.004

Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters* 103 (2009), 150502. Issue 15. https://doi.org/10.1103/PhysRevLett.103.150502

Charles Antony Richard Hoare. 1969. An axiomatic basis for computer programming. *Communications of the ACM* 12, 10 (1969), 576–580.

Xiaodong Jia, Andre Kornell, Bert Lindenhovius, Michael Mislove, and Vladimir Zamdzhiev. 2022. Semantics for Variational Quantum Programming. *Proc. of the ACM on Programming Languages* 6, POPL, Article 26 (2022), 31 pages. https://doi.org/10.1145/3498687

Xiaodong Jia, Bert Lindenhovius, Michael W. Mislove, and Vladimir Zamdzhiev. 2021. Commutative Monads for Probabilistic Programming Languages. In *LICS 2021*. IEEE, 1–14. https://doi.org/10.1109/LICS52264.2021.9470611

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. In *ESOP 2016 (LNCS, Vol. 9632)*. Springer, 364–389. https://doi.org/10.1007/978-3-662-49498-1_15

Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2017. A weakest pre-expectation semantics for mixed-sign expectations. In *LICS 2017*. IEEE, 1–12. https://doi.org/10.1109/LICS.2017.8005153

Klaus Keimel and Gordon D. Plotkin. 2017. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science* 13, 1 (2017). https://doi.org/10.23638/LMCS-13(1:2)2017

Dexter Kozen. 1985. A probabilistic pdl. *Journal of Computer and System Sciences* 30, 2 (1985), 162–178.

Thomas Leventis and Michele Pagani. 2019. Strong Adequacy and Untyped Full-Abstraction for Probabilistic Coherence Spaces. In *Proc. of FoSSaCS 2020 (LNCS, Vol. 11425)*. Springer, 365–381. https://doi.org/10.1007/978-3-030-17127-8_21

Junyi Liu, Li Zhou, Gilles Barthe, and Mingsheng Ying. 2022. Quantum Weakest Preconditions for Reasoning about Expected Runtimes of Quantum Programs. arXiv:1911.12557

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, refinement and proof for probabilistic systems*. Springer Science & Business Media.

Fabian Meyer, Marcel Hark, and Jürgen Giesl. 2021. Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes. In *Proc. of TACAS 2021 (LNCS, Vol. 12651)*. Springer, 250–269. https://doi.org/10.1007/978-3-030-72016-2_14

Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded expectations: resource analysis for probabilistic programs. *ACM SIGPLAN*

Notices 53, 4 (2018), 496–512. https://doi.org/10.1145/3296979.3192394

Michael A. Nielsen. 2004. Optical Quantum Computation Using Cluster States. *Physical Review Letters* 93 (2004), 040503. Issue 4. https://doi.org/10.1103/PhysRevLett.93.040503

Federico Olmedo and Alejandro Díaz-Caro. 2020. Runtime Analysis of Quantum Programs: A Formal Approach. In *PLanQC 2020*. arXiv:1911.11247

Adam Paetznick and Krysta M Svore. 2013. Repeat-Until-Success: Non-deterministic decomposition of single-qubit unitaries. arXiv:1311.1074

Romain Péchoux, Simon Perdrix, Mathys Rennela, and Vladimir Zamdzhiev. 2020. Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory. In *Proc. of FoSSaCS 2020 (LNCS, Vol. 12077)*. Springer, 562–581. https://doi.org/10.1007/978-3-030-45231-5_29

Mathys Rennela. 2020. Convexity and Order in Probabilistic Call-by-Name FPC. *Logical Methods in Computer Science* 16, 4 (2020). https://doi.org/10.23638/LMCS-16(4:10)2020

Peter Selinger. 2004. Towards a quantum programming language. *Mathematical Structures in Computer Science* 14, 4 (2004), 527–586. https://doi.org/10.1017/S0960129504004256

Peter W. Shor. 1999. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Review* 41, 2 (1999), 303–332. https://doi.org/10.1137/S0036144598347011

Peixin Wang, Hongfei Fu, Amir K. Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost Analysis of Nondeterministic Probabilistic Programs. In *Proc. of PLDI 2019*. ACM, 204–220. https://doi.org/10.1145/3314221.3314581