Quantum Relational Hoare Logic with Expectations

Yangjia Li
University of Tartu, Estonia
SKLCS, Institute of Software, CAS, Beijing, China

Dominique Unruh
University of Tartu, Estonia

Abstract

We present a variant of the quantum relational Hoare logic from (Unruh, POPL 2019) that allows us to use "expectations" in pre- and postconditions. That is, when reasoning about pairs of programs, our logic allows us to quantitatively reason about how much certain pre-/postconditions are satisfied that refer to the relationship between the programs inputs/outputs.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Hoare logic; Theory of computation → Program verification

Keywords and phrases Quantum cryptography, Hoare logics, formal verification

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.136

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/1903.08357

Funding By the Air Force Office of Scientific Research through the project "Verification of quantum cryptography" (AOARD Grant FA2386-17-1-4022). By the project "Research and preparation of an ERC grant application on Certified Quantum Security" (MOBERC12). By the ERC consolidator grant CerQuS (819317). By the PRG team grant "Secure Quantum Technology" (PRG946) from the Estonian Research Council.
Yangjia Li: By the National Natural Science Foundation of China (Grant No: 61872342).
Dominique Unruh: By institutional research funding IUT2-1 of the Estonian Ministry of Education and Research. By the Estonian Centre of Exellence in IT (EXCITE) funded by the ERDF.

Acknowledgements We thank Gilles Barthe, Tore Vincent Carstens, and Justin Hsu for valuable discussions.

1 Introduction

Relational Hoare logics (RHL) are logics that allow us to reason about the relationship between two programs. Roughly speaking, they can express facts like "if the variable $\mathbf{x}$ in program $\mathfrak{c}$ is equal to $\mathbf{x}$ in program $\mathfrak{d}$, then after executing $\mathfrak{c}$ and $\mathfrak{d}$, respectively, the content of variable $\mathbf{y}$ in program $\mathfrak{c}$ is greater than that of $\mathbf{y}$ in $\mathfrak{d}$." RHL was introduced in the deterministic case by [6], and generalized to probabilistic programs by [4] (pRHL) and to quantum programs by [17] (qRHL). RHLs have proven especially useful in the context of verification of cryptographic schemes. For example, the CertiCrypt tool [4, 3] and its successor EasyCrypt [2, 1] use pRHL to create formally verified cryptographic proofs. And [16] implements a tool for verifying quantum cryptographic proofs based on qRHL.

On the other hand, "normal" (i.e., not relational) quantum Hoare logics have been developed in the quantum setting, starting with the predicate transformers from [10], see [11, 18, 7, 12]. Out of these, [10, 11, 18] use "expectations" instead of "predicates" for the pre- and postconditions of the Hoare judgments. To understand the difference, consider the case of classical probabilistic programs. Here, a predicate is (logically equivalent to) a set of program states (and a program state is a function from variables to values). In contrast, an

expectation is a function from program states to real numbers, basically assigning a value to each program state. Probabilistic Hoare logic with expectations, implicit in [13], uses expectations as the pre- and postconditions of a Hoare judgment. Then, roughly speaking, the preexpectation tells us what the expected value of the postexpectation is after running the program. This can be used to express much more fine-grained properties of probabilistic programs, giving quantitative guarantees about their probabilistic behavior, instead of just qualitative (a certain final state can or cannot occur). As [10] showed, the same approach can be used for quantum programs. Here, an expectation is modeled by a self-adjoint operator A on the space of all program states. The "value" of a given program state ρ is then computed as the trace $\operatorname{tr}\mathsf{A}\rho$. While at the first glance not as obvious as the meaning of classical expectations, this formalism has nice mathematical properties and is also equivalent to taking the expectation value of the outcome of a real-valued measurement. By using this approach, [10, 11, 18] can express more fine-grained judgments about quantum programs, by not just expressing which final states are possible, but also with what probabilities.

Yet, qRHL [17] did not follow this approach (only mentioning it as possible future work). As a consequence, qRHL does not enable as fine-grained reasoning about probabilities as the non-relational quantum Hoare logics. On the other hand, the non-relational quantum Hoare logics do not allow us to reason about the relationship between programs.

In this work, we combine the best of two worlds. We present a variant of qRHL, expectation-qRHL, that reasons about pairs of programs, and at the same time supports expectations as the pre- and postconditions, thus being as expressive as the calculi from [10, 11, 18] when it comes to the probabilistic behavior of the programs.

Related work. The relevant prior work has already been discussed above. Concurrently and independently, [5] presented a different formalization of expectation-qRHL. (The first versions on arXiv appeared within two months of each other.) The biggest difference is the definition of couplings which in our setting are separable quantum states, and in their setting nonseparable quantum states. Therefore, the soundness proofs are totally different in [5] and in the present paper, even for the same rules. As a consequence, we can avoid having to reason about judgments with side-conditions, making compositional reasoning about more complex programs much easier.

Organization. In Section 2 we introduce notation and preliminaries, including the concept of expectations. In Section 3 we give syntax and semantics of the imperative quantum programming language that we study. In Section 4 we give the definition of expectation-qRHL. In Section 5, we present sound rules for reasoning about expectation-qRHL judgments. And in Section 6, we analyze the quantum Zeno effect as an example of using our logic.

In the full version, we give a detailed comparison of our logic with [5] and full proofs of our results.

2 Preliminaries: Variables, Memories, and Predicates

In this section, we introduce some fundamental concepts and notations needed for this paper, and recap some of the needed quantum background as we go along. When introducing some notation X, the place of definition is marked like this: X. For further mathematical background we recommend [8, 9], and for an introduction to quantum mechanics [15].

Variables. Before we introduce the syntax and semantics of programs, we first need to introduce some basic concepts. A *variable* is described by a variable name $\mathbf{x}, \mathbf{y}, \mathbf{z}$ that identifies the variable, and a nonempty type T. The *type* of $\mathbf{x}$ is simply the nonempty set of all (classical) values the variable can take. E.g., a variable might have type $\{0,1\}$, or $\mathbb{N}$.[1] Lists or sets of variables will be denoted $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$. Given a list $\mathbf{X} = \mathbf{x}_1 \dots \mathbf{x}_n$ of variables, we say its *type* is $T_1 \times \cdots \times T_n$ if T_i is the type of $\mathbf{x}_i$. We write $\mathbf{XY}$ for the concatenation/disjoint union of lists/sets of variables $\mathbf{X}, \mathbf{Y}$.

Memories and quantum states. An *assignment* assigns to each variable a classical value. Formally, for a set $\mathbf{X}$, the *assignments over* $\mathbf{X}$ are all functions m with domain $\mathbf{X}$ such that: for all $\mathbf{x} \in \mathbf{X}$ with type $T_{\mathbf{x}}$, $\mathsf{m}(\mathbf{x}) \in T_{\mathbf{x}}$. That is, assignments can represent the content of classical memories.

To model quantum memories, we simply consider superpositions of assignments: A *(pure) quantum memory* is a superposition of assignments. Formally, $\ell^2[\mathbf{X}]$, the set of all quantum memories over $\mathbf{X}$, is the Hilbert space with basis[2] $\{|\mathsf{m}\rangle\}_{\mathsf{m}}$ where m ranges over all assignments over $\mathbf{X}$. Here $|\mathsf{m}\rangle$ simply denotes the basis vector labeled m. We often write $|\mathsf{m}\rangle_{\mathbf{X}}$ to stress which space we are talking about. We call a quantum memory ψ *normalized* iff $\|\psi\| = 1$. Intuitively, a normalized quantum memory over $\mathbf{X}$ represents a state a quantum computer with variables $\mathbf{X}$ could be in. We also consider quantum states over arbitrary sets X (as opposed to sets of assignments). Namely, $\ell^2(X)$ denotes the Hilbert space with orthonormal basis $\{|x\rangle\}_{x\in X}$. (In that notation, $\ell^2[\mathbf{X}]$ is simply $\ell^2(A)$ where A is the set of all assignments on $\mathbf{X}$.) Normalized elements of $\ell^2[\mathbf{X}]$ represent quantum states.

We often treat elements of $\ell^2(T)$ and $\ell^2[\mathbf{X}]$ interchangeably if T is the type of $\mathbf{X}$ since there is a natural isomorphism between those spaces.

In many situations, we additionally need an additional symbol $\bot$ that denotes that a memory is not available because the program did not terminate. A *quantum $\bot$-memory over* $\mathbf{X}$ is an element of $\ell^2[\mathbf{X}]^\bot := \ell^2(A \cup \{\bot\})$ where A is the set of all assignments on $\mathbf{X}$. That is, a quantum $\bot$-memory is a superposition between a quantum memory and $|\bot\rangle$.

The tensor product $\otimes$ combines two quantum states $\psi \in \ell^2(X), \phi \in \ell^2(Y)$ into a joint system $\psi \otimes \phi \in \ell^2(X \times Y)$. In the case of quantum memories ψ, ϕ over $\mathbf{X}, \mathbf{Y}$, respectively, $\psi \otimes \phi \in \ell^2[\mathbf{XY}]$. (And in this case, $\psi \otimes \phi = \phi \otimes \psi$ since we are composing "named" systems.)

We will need to consider entangled pairs of memories. Specifically, a *quantum bimemory* over $\mathbf{X}_1, \mathbf{X}_2$ is an element of $\ell^2[\mathbf{X}_1] \otimes \ell^2[\mathbf{X}_2] = \ell^2[\mathbf{X}_1\mathbf{X}_2]$. Similarly, a *quantum $\bot$-bimemory* is an element of $\ell^2[\mathbf{X}_1]^\bot \otimes \ell^2[\mathbf{X}_2]^\bot$, i.e., a tensor product of two quantum $\bot$-memories. (Note: "one-sided-$\bot$" states such as $|\mathsf{m}\rangle \otimes |\bot\rangle$ are included in this.) For clarity, we often write $|\bot_1\rangle, |\bot_2\rangle$ instead of $\bot$ to emphasize whether we are talking about elements of $\ell^2[\mathbf{X}_1]^\bot$ or $\ell^2[\mathbf{X}_2]^\bot$.

For a vector (or operator) a, we write a^* for its adjoint. (In the finite dimensional case, the adjoint is simply the conjugate transpose of a vector/matrix. The literature also knows the notation $a^\dagger$.) The adjoint of a vector $|x\rangle$ is also written as $\langle x|$. We abbreviate $\mathsf{proj}(\psi) := \psi\psi^*$. This is the projector onto ψ when $\|\psi\| = 1$.

[1] We stress that we do not assume that the type is a finite or even a countable set. Consequently, the Hilbert spaces considered in this paper are not necessarily finite dimensional or even separable. However, all results can be informally understood by thinking of all sets as finite and hence of all Hilbert spaces as $\mathbb{C}^N$ for suitable $N \in \mathbb{N}$.

[2] When we say "basis", we always mean an orthonormal Hilbert-space basis.

Mixed quantum memories. In many situations, we need to model probabilistic quantum states (e.g., a quantum state that is $|0\rangle$ with probability $\frac{1}{2}$ and $|1\rangle$ with probability $\frac{1}{2}$). This is modeled using *mixed states* (a.k.a. *density operators*). Having normalized state ψ_i with probability p_i is represented by the operator $\rho := \sum_i p_i \mathsf{proj}(\psi_i)$.[3] In particular, $\mathsf{proj}(\psi)$ is the density operator of a pure quantum state ψ. Then ρ encodes all observable information about the distribution of the quantum state (that is, two distributions of quantum states have the same ρ iff they cannot be distinguished by any physical process). And $\operatorname{tr}\rho$ is the total probability $\sum_i p_i$. Note that we do not formally impose the condition $\operatorname{tr}\rho = 1$ or $\operatorname{tr}\rho \leq 1$ unless explicitly specified. We call a mixed state *normalized* iff $\operatorname{tr}\rho = 1$. We will often need to consider mixed states of quantum memories (i.e., mixed states with underlying Hilbert space $\ell^2[\mathbf{X}]$). We call them *mixed (quantum) memories* over $\mathbf{X}$. Analogously, we define *mixed bimemories* and *mixed $\bot$-bimemories* as mixed states of quantum ($\bot$-)bimemories.

Let $P_\bot := \mathsf{proj}(|\bot\rangle)$ and $P_{\not\bot} := \mathsf{id} - P_\bot$. We can easily access the terminating and non-terminating part of a mixed $\bot$-memory: $\downarrow_\bot(\rho) := P_\bot \rho P_\bot$ and $\downarrow_{\not\bot}(\rho) := P_{\not\bot}\rho P_{\not\bot}$ are the memory ρ after measuring that we have termination or do not have termination, respectively.

For a mixed ($\bot$-)bimemory ρ over $\mathbf{X}_1\mathbf{X}_2$ the *partial trace* $\operatorname{tr}_i \rho$ ($i = 1, 2$) is the result of throwing away the left/right memory (i.e., it is a mixed memory over $\mathbf{X}_i$). Formally, tr_i is defined as the continuous linear function satisfying $\operatorname{tr}_1(\sigma \otimes \tau) := \tau \cdot \operatorname{tr}\sigma$, $\operatorname{tr}_2(\sigma \otimes \tau) := \sigma \cdot \operatorname{tr}\tau$.

A mixed memory ρ is $(\mathbf{X}, \mathbf{Y})$-*separable* (i.e., not entangled between $\mathbf{X}$ and $\mathbf{Y}$) iff it can be written as $\rho = \sum_i \rho_i \otimes \rho'_i$ for mixed memories ρ_i, ρ'_i over $\mathbf{X}, \mathbf{Y}$, respectively. (Potentially infinite sum.) When $\mathbf{X}, \mathbf{Y}$ are clear from the context, we simply say *separable*.

In this paper, when we write infinite sums of operators, convergence is always with respect to the trace norm: $\|A\|_{\operatorname{tr}} := \operatorname{tr}\sqrt{AA^\dagger}$. (In the finite-dimensional case, the choice of norm is irrelevant since all norms are equivalent then.)

Operations on quantum states. An operation in a closed quantum system is modeled by an isometry U on $\ell^2(X)$.[4] If we apply such an operation on a mixed state ρ, the result is $U\rho U^*$. In particular, denote by id the identity operation, i.e. $\mathsf{id}\psi = \psi$ for all pure states ψ in this space.

An operator A on $\ell^2[\mathbf{X}]$ can be interpreted as an operator on $\ell^2[\mathbf{X}]^\bot$ by setting $A|\bot\rangle := 0$. To avoid confusion, we often write $A \oplus 0_\bot$ for the operator on $\ell^2[\mathbf{X}]^\bot$. Similarly, an operator A on $\ell^2[\mathbf{X}_1] \otimes \ell^2[\mathbf{X}_2]$ can be seen as an operator on $\ell^2[\mathbf{X}_1]^\bot \otimes \ell^2[\mathbf{X}_2]^\bot$, we write $A \oplus 0_{\bot\bot}$ for the operator on $\ell^2[\mathbf{X}_1]^\bot \otimes \ell^2[\mathbf{X}_2]^\bot$.

Most often, isometries will occur in the context of operations that are performed on a single variable or list of variables, i.e., an isometry U on $\ell^2[\mathbf{X}]$. Then U can also be applied to $\ell^2[\mathbf{Y}]$ with $\mathbf{Y} \supseteq \mathbf{X}$: we identify U with $U \otimes \mathsf{id}_{\mathbf{Y}\setminus\mathbf{X}}$. Furthermore, if $\mathbf{X}$ has type T, then an isometry U on $\ell^2(T)$ can be seen as an isometry on $\ell^2[\mathbf{X}]$ since we identify $\ell^2(T)$ and $\ell^2[\mathbf{X}]$. If we want to make $\mathbf{X}$ explicit, we write $U\,\mathbf{on}\,\mathbf{X}$ for the isometry U on $\ell^2[\mathbf{Y}]$. For example, if U is a 2×2-matrix and $\mathbf{x}$ has type bit, then $U\,\mathbf{on}\,\mathbf{x}$ can be applied to quantum memories over $\mathbf{xy}$, acting on $\mathbf{x}$ only. This notation is not limited to isometries, of course, but applies to other operators, too. (By *operator* we always mean a bounded linear operator in this paper.)

---

[3] Mathematically, density operators are positive Hermitian trace-class operators on $\ell^2(X)$. The requirement "trace-class" ensures that the trace exists and can be ignored in the finite-dimensional case.

[4] That is, a norm-preserving linear operation. Often, one models quantum operations as unitaries instead because in the finite-dimensional case an isometry is automatically unitary. However, in the infinite-dimensional case, unitaries are unnecessarily restrictive. Consider, e.g., the isometry $|i\rangle \mapsto |i+1\rangle$ with $i \in \mathbb{N}$ which is a perfectly valid quantum operation but not a unitary.

In slight overloading of notation, we also write $U \mathbf{on} \mathbf{X}$ for U acting on $\bot$-memories, where $(U \mathbf{on} \mathbf{X})|\bot\rangle = 0$. (That is, $U \mathbf{on} \mathbf{X}$ is short for the more precise $(U \mathbf{on} \mathbf{X}) \oplus 0_{\bot}$.) We also write $U \mathbf{on} \mathbf{X}_1$ for U acting on $\bot$-bimemories. In this case, we simply have $U \mathbf{on} \mathbf{X}_1 := (U \mathbf{on} \mathbf{X}_1) \otimes \mathsf{id}$. In particular, $(U \mathbf{on} \mathbf{X}_1)(|\mathsf{m}\rangle \otimes |\bot\rangle) = (U \mathbf{on} \mathbf{X}_1)|\mathsf{m}\rangle \otimes |\bot\rangle$ but $(U \mathbf{on} \mathbf{X}_1)(|\bot\rangle \otimes |\mathsf{m}\rangle) = 0$. Analogously for $U \mathbf{on} \mathbf{X}_2$.

We will use only binary measurements in this paper. A *binary measurement* M on $\ell^2[\mathbf{X}]$ has outcomes $\mathsf{true}, \mathsf{false}$ and is described by two bounded operators M_{true}, M_{false} on $\ell^2[\mathbf{X}]$ that satisfy $M^*_{\mathsf{true}} M_{\mathsf{true}} + M^*_{\mathsf{false}} M_{\mathsf{false}} = \mathsf{id}$, its *Krauss operators*. Given a mixed memory ρ, the probability of measurement outcome t is $p_t := \operatorname{tr} M_t \rho M_t^*$, and the post-measurement state is $M_t \rho M_t^* / p_t$.

Expectations. In this work, we will use expectations as pre- and postconditions in Hoare judgments. The idea of using expectations originated in [13] for reasoning about (classical) probabilistic programs. Intuitively, an expectation is a quantitative predicate, that is for any memory (or bimemory, in our case), it does not tell us whether the memory satisfies the predicate but *how much* it satisfies the predicate. Thus, classically, an expectation is simply a function from assignments to reals. By analogy, in the quantum setting, one might want to define expectations, e.g., as functions f from quantum bimemories to reals (i.e., an expectation would be a function $\ell^2[\mathbf{X}] \to \mathbb{R}_{\geq 0}$). However, such expectations might behave badly, for example, it is not clear how to compute the expected value $f(\psi)$ for a random ψ if the distribution of ψ is given in terms of a density operator. A better approach was introduced by [10]. Following their approach, we define an *expectation* as a positive operator A on quantum bimemories.[5] (We use letters $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ for expectations in this paper.) This expectation then assigns the value $\psi^* \mathsf{A} \psi$ to the quantum memory ψ (equivalently, $\operatorname{tr} \mathsf{A}\, \mathsf{proj}(\psi)$). To understand this, it is best to first look at the special case where A is a projector. Then $\psi^* \mathsf{A} \psi = 1$ iff ψ is in the image of A, and $\psi^* \mathsf{A} \psi = 0$ iff ψ is orthogonal to the image of A. Such an A is basically a predicate (by outputting 1 for states that satisfy the predicate). Of course, states that neither satisfy the predicate nor are orthogonal to it will output a value between 0 and 1. Any expectation A can be written as $\sum_i p_i \mathsf{A}_i$ with projectors A_i. Thus, A would give p_i "points" for satisfying the predicate A_i. In this respect, expectations in the quantum setting are similar to classical ones: classical expectations give a certain amount of "points" for each possible classical input.

The nice thing about this formalism is that, given a density operator $\rho = \sum p_i \mathsf{proj}(\psi_i)$, we can easily compute the expected value of the expectation A. More precisely, the expected value of $\psi^* \mathsf{A} \psi = \operatorname{tr} \mathsf{A} \mathsf{proj}(\psi)$ with $\psi := \psi_i$ with probability p_i. That expected value is $\sum p_i \operatorname{tr} \mathsf{A} \mathsf{proj}(\psi_i) = \operatorname{tr} \mathsf{A} (\sum p_i \mathsf{proj}(\psi_i)) = \operatorname{tr} \mathsf{A} \rho$. This shows that we can evaluate how much a density operator satisfies the expectation A by just computing $\operatorname{tr} \mathsf{A} \rho$. This formula will be the basis for our definitions!

Analogously, we define $\bot$-*expectations* on quantum $\bot$-bimemories. However, we add one restriction: The value of a $\bot$-expectation should not change if we measure whether the $\bot$-bimemory is in $|\bot\rangle$ or not. Formally, a $\bot$-expectation is a positive operator on quantum $\bot$-bimemories that is invariant under $\mathcal{E}_{\bot} \otimes \mathcal{E}_{\bot}$ where $\mathcal{E}_{\bot}(\rho) := P_{\bot} \rho P_{\bot} + P_{\not\bot} \rho P_{\not\bot}$. ($\mathcal{E}_{\bot}$ corresponds to measuring and forgetting whether a given mixed $\bot$-memory is $|\bot\rangle$ or not.) Note that for an expectation A, the operator $\mathsf{A} \oplus 0_{\bot\bot}$ is a $\bot$-expectation. We can thus see expectations as special cases of $\bot$-expectations.

[5] Recall from page 4 that operators are always bounded in our context. This means that A is bounded, too. This means that the values that an expectation A can assign to states are between 0 and B for some finite B.

A very simple example of an expectation would be the matrix $\mathsf{A} := \begin{pmatrix} 1 & \\ & \frac{1}{2} \end{pmatrix}$ that assigns 1 to $|0\rangle$, and $\frac{1}{2}$ to $|1\rangle$. And given the density operator $\rho = \frac{1}{2}\mathsf{id}$ (representing a uniform qubit), $\operatorname{tr} \mathsf{A}\rho = \frac{3}{4}$ are intuitively expected.

Quantum equality. In [17], a specific predicate $\mathbf{X}_1 \equiv_\mathfrak{q} \mathbf{X}_2$ was introduced to describe the fact that two quantum variables (or list of quantum variables) have the same state. Formally, $\mathbf{X}_1 \equiv_\mathfrak{q} \mathbf{X}_2$ is the subspace consisting of all quantum memories in $\ell^2[\mathbf{X}_1\mathbf{X}_2]$ that are invariant under $\mathsf{SWAP}_{\mathbf{X}_1\mathbf{X}_2}$, the unitary that swaps variables $\mathbf{X}_1$ and $\mathbf{X}_2$.[6] Or equivalently, $\mathbf{X}_1 \equiv_\mathfrak{q} \mathbf{X}_2$ denotes the subspace spanned by all quantum memories of the form $\phi \otimes \phi$ with $\phi \in \ell^2[\mathbf{X}_1] = \ell^2[\mathbf{X}_2]$. We write $\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{X}_1\mathbf{X}_2$ for the projector onto $\mathbf{X}_1 \equiv_\mathfrak{q} \mathbf{X}_2$.

3 Quantum programs

Syntax. We will now define a small imperative quantum language.[7] The set of all programs is described by the following syntax:

$$\mathfrak{c},\mathfrak{d} ::= \underline{\mathbf{apply}}\ U\ \underline{\mathbf{to}}\ \mathbf{X} \mid \mathbf{X} \leftarrow \psi \mid \underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}\ \underline{\mathbf{else}}\ \mathfrak{d} \mid \underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c} \mid \mathfrak{c};\mathfrak{d} \mid \underline{\mathbf{skip}} \mid \underline{\mathbf{abort}}$$

Here $\mathbf{X}$ is a list of variables and U an isometry on $\ell^2[\mathbf{X}]$, $\psi \in \ell^2[\mathbf{X}]$ a normalized state, and M is a binary measurement on $\ell^2[\mathbf{X}]$. (There are no fixed sets of allowed U and ψ, any isometry/state that we can describe can be used here).[8]

Intuitively, $\underline{\mathbf{apply}}\ U\ \underline{\mathbf{to}}\ \mathbf{X}$ means that the operation U is applied to the quantum variables $\mathbf{X}$. E.g., $\underline{\mathbf{apply}}\ H\ \underline{\mathbf{to}}\ \mathbf{x}$ would apply the Hadamard gate to the variable $\mathbf{x}$ (we assume that H denote the Hadamard matrix). It is important that we can apply U to several variables $\mathbf{X}$ simultaneously, otherwise no entanglement between variables can ever be produced.

The program $\mathbf{X} \leftarrow \psi$ initializes the variables $\mathbf{X}$ with the quantum state ψ. The program $\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}\ \underline{\mathbf{else}}\ \mathfrak{d}$ will measure the variables $\mathbf{X}$ with the measurement M, and, if the outcome is true, execute $\mathfrak{c}$, otherwise execute $\mathfrak{d}$.

The program $\underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c}$ measures $\mathbf{X}$, and if the outcome is true, it executes $\mathfrak{c}$. This is repeated until the outcome is false.

Finally, $\mathfrak{c};\mathfrak{d}$ executes $\mathfrak{c}$ and then $\mathfrak{d}$. And $\underline{\mathbf{skip}}$ does nothing. We will always implicitly treat ";" as associative and $\underline{\mathbf{skip}}$ as its neutral element. $\underline{\mathbf{abort}}$ never terminates.

Semantics. The *denotational semantics* of our programs $\mathfrak{c}$ are represented as completely positive trace-reducing maps $[\![\mathfrak{c}]\!]$ on the mixed memories over $\mathbf{X}^{\mathbf{all}}$, defined by recursion on the structure of the programs. Here $\mathbf{X}^{\mathbf{all}}$ is a fixed set of program variables, and we will assume that all variables under consideration are contained in this set. The obvious cases are $[\![\underline{\mathbf{skip}}]\!] := \mathsf{id}$ and $[\![\mathfrak{c};\mathfrak{d}]\!] := [\![\mathfrak{d}]\!] \circ [\![\mathfrak{c}]\!]$ and $[\![\underline{\mathbf{abort}}]\!](\rho) := 0$. And application of an isometry U is also fairly straightforward given the syntactic sugar introduced above: $[\![\underline{\mathbf{apply}}\ U\ \underline{\mathbf{to}}\ \mathbf{X}]\!](\rho) := (U\,\mathbf{on}\,\mathbf{X})\rho(U\,\mathbf{on}\,\mathbf{X})^*$. (The notation $U\,\mathbf{on}\,\mathbf{X}$ was introduced on page 4.)

---

[6] That is, $\mathsf{SWAP}_{\mathbf{X}_1\mathbf{X}_2}(\psi \otimes \phi) = \phi \otimes \psi$ for $\psi \in \ell^2[\mathbf{X}_1]$, $\phi \in \ell^2[\mathbf{X}_2]$.

[7] Very similar to [18], except that we replace their case-statement by an if-statement and allow initialization with arbitrary states instead of just $|0\rangle$.

[8] We will assume throughout the paper that all programs satisfy those well-typedness constraints. In particular, rules may implicitly impose type constraints on the variables and constants occurring in them by this assumption.

Initialization of quantum variables is slightly more complicated: $\mathbf{X} \leftarrow \psi$ initializes the variables $\mathbf{X}$ with ψ, which is the same as removing $\mathbf{X}$, and then creating a new variable $\mathbf{X}$ with content ψ. Removing $\mathbf{X}$ is done by the operation $\mathrm{tr}_{\mathbf{X}}$ (partial trace, see page 4). And creating new variables $\mathbf{X}$ in state ψ is done by the operation $\otimes\mathsf{proj}(\psi)$. Thus we define $[\![\mathbf{X} \leftarrow \psi]\!](\rho) := \mathrm{tr}_{\mathbf{X}}\, \rho \otimes \mathsf{proj}(\psi)$.

The if-command first performs a measurement and then branches depending on the outcome. We then have that the state after measurement (without renormalization) is $(M_t \,\mathbf{on}\, \mathbf{X})\rho(M_t \,\mathbf{on}\, \mathbf{X})^*$ for outcome $t = \mathsf{true}, \mathsf{false}$. Then $\mathfrak{c}$ or $\mathfrak{d}$ is applied to that state and the resulting states are added together to get the final mixed state. Altogether:

$$[\![\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}\ \underline{\mathbf{else}}\ \mathfrak{d}]\!](\rho) := [\![\mathfrak{c}]\!]\big(\downarrow_{\mathsf{true}}(\rho)\big) + [\![\mathfrak{d}]\!]\big(\downarrow_{\mathsf{false}}(\rho)\big)$$
$$\text{where} \qquad \downarrow_t(\rho) := (M_t \,\mathbf{on}\, \mathbf{X})\rho(M_t \,\mathbf{on}\, \mathbf{X})^*$$

While-commands are modeled similarly: In an execution of a while statement, we have $n \geq 0$ iterations of "measure with outcome true and run $\mathfrak{c}$" (which applies $[\![\mathfrak{c}]\!] \circ \downarrow_{\mathsf{true}}$ to the state), followed by "measure with outcome false" (which applies $\downarrow_{\mathsf{false}}$ to the state). Adding all those branches up, we get the definition:

$$[\![\underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c}]\!](\rho) := \sum_{n=0}^{\infty} \downarrow_{\mathsf{false}}\big(([\![\mathfrak{c}]\!] \circ \downarrow_{\mathsf{true}})^n(\rho)\big)$$

We call a program $\mathfrak{c}$ *terminating* iff $\mathrm{tr}[\![\mathfrak{c}]\!](\rho) = \mathrm{tr}\,\rho$ for all ρ.

Semantics with explicit non-termination. $[\![\mathfrak{c}]\!]$ is not trace-preserving if $\mathfrak{c}$ is not terminating. For technical reasons, we will need a variant of this function that is trace-preserving. This can be achieved by outputting a mixed state that has an explicit state $\mathsf{proj}(|\bot\rangle)$ to denote non-termination. This semantic function $[\![\mathfrak{c}]\!]^\bot$ takes mixed $\bot$-memories to mixed $\bot$-memories and can be easily derived from $[\![\mathfrak{c}]\!]$:

$$[\![\mathfrak{c}]\!]^\bot(\rho) := [\![\mathfrak{c}]\!](\downarrow_{\not\bot}(\rho)) + \big(\mathrm{tr}\,\rho - \mathrm{tr}[\![\mathfrak{c}]\!](\downarrow_{\not\bot}(\rho))\big)\mathsf{proj}(|\bot\rangle).$$

($P_\bot, P_{\not\bot}$ are defined on page 4.) Operationally, $[\![\mathfrak{c}]\!]^\bot$ first measures if the state is $\bot$. If so, nothing happens. Otherwise, $\mathfrak{c}$ is applied. If $\mathfrak{c}$ does not terminate, the final output memory is set to be $\bot$. $[\![\mathfrak{c}]\!]^\bot$ is easily seen to be trace-preserving. Moreover, we have the composition property $[\![\mathfrak{c};\mathfrak{d}]\!]^\bot = [\![\mathfrak{d}]\!]^\bot \circ [\![\mathfrak{c}]\!]^\bot$, since

$$\begin{aligned}
&[\![\mathfrak{d}]\!]^\bot([\![\mathfrak{c}]\!]^\bot(\rho)) = [\![\mathfrak{d}]\!]^\bot\,[[\![\mathfrak{c}]\!](\rho_{\not\bot}) + (\mathrm{tr}\,\rho - \mathrm{tr}[\![\mathfrak{c}]\!](\rho_{\not\bot}))\mathsf{proj}(|\bot\rangle)] \\
&= [\![\mathfrak{d}]\!]([\![\mathfrak{c}]\!](\rho_{\not\bot})) + \big[\mathrm{tr}[\![\mathfrak{c}]\!]^\bot(\rho) - \mathrm{tr}[\![\mathfrak{d}]\!]([\![\mathfrak{c}]\!](\rho_{\not\bot}))\big]\,\mathsf{proj}(|\bot\rangle) \\
&= [\![\mathfrak{c};\mathfrak{d}]\!](\rho_{\not\bot}) + (\mathrm{tr}\,\rho - \mathrm{tr}[\![\mathfrak{c};\mathfrak{d}]\!](\rho_{\not\bot}))\mathsf{proj}(|\bot\rangle) = [\![\mathfrak{c};\mathfrak{d}]\!]^\bot(\rho).
\end{aligned}$$

4 qRHL with expectations

Defining the logic. We now present our definition of expectation-qRHL. We follow the approach from [17] to use separable couplings to describe the relationship between programs. A *coupling* between two mixed states ρ_1 and ρ_2 is a mixed state ρ that has ρ_1 and ρ_2 as marginals. (That is, $\mathrm{tr}_{\mathbf{X}_2}\,\rho = \rho_1$ and $\mathrm{tr}_{\mathbf{X}_1}\,\rho = \rho_2$ if ρ_1, ρ_2 are over $\mathbf{X}_1, \mathbf{X}_2$, respectively.) This is analogous to probabilistic couplings: a coupling of distributions μ_1, μ_2 is a distribution μ with marginals μ_1, μ_2. Note that couplings trivially always exist if ρ_1 and ρ_2 have the same trace (namely, $\rho := \rho_1 \otimes \rho_2 / \mathrm{tr}\,\rho_1$). Couplings become interesting when we put additional

constraints on the state ρ. For example, if we require the support of ρ to be in the subspace $C := \mathrm{span}\{|00\rangle, |11\rangle\}$, then $\rho_1 = \mathsf{proj}(|0\rangle)$ and $\rho_2 = \mathsf{proj}(|0\rangle)$ have a coupling (namely, $\rho = \mathsf{proj}(|00\rangle)$), as do $\rho_1 = \mathsf{proj}(|1\rangle)$ and $\rho_2 = \mathsf{proj}(|1\rangle)$ (namely, $\rho = \mathsf{proj}(|11\rangle)$), but not $\rho_1 = \mathsf{proj}(|0\rangle)$ and $\rho_2 = \mathsf{proj}(|1\rangle)$. Things become particularly interesting when ρ_1, ρ_2 are not pure states. E.g., $\rho_1 = \frac{1}{2}\mathsf{proj}(|0\rangle) + \frac{1}{2}\mathsf{proj}(|1\rangle)$ and $\rho_2 = \frac{1}{2}\mathsf{proj}(|0\rangle) + \frac{1}{2}\mathsf{proj}(|1\rangle)$ have such a coupling as well (namely, $\rho = \frac{1}{2}\mathsf{proj}(|00\rangle) + \frac{1}{2}\mathsf{proj}(|11\rangle)$ but $\rho := \rho_1 \otimes \rho_2$ is not a coupling with support in C).

Thus, a subspace such as C can be seen as a predicate describing the relationship of ρ_1, ρ_2. The states ρ_1, ρ_2 satisfy C iff there is a coupling with support in C. This idea leads to the following tentative definition of qRHL:

▶ **Definition 1** (qRHL, tentative, without expectations). *For subspaces A, B (i.e., spaces of quantum memories over $\mathbf{X}_1^{\mathbf{all}}\mathbf{X}_2^{\mathbf{all}}$), $\{A\}\,\mathfrak{c} \sim \mathfrak{d}\,\{B\}$ holds iff for any ρ_1, ρ_2 that have a coupling with support in A, the final states $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$ have a coupling with support in B.*

However, it was noticed in [17] that the definition becomes easier to handle if we impose another condition on the couplings. Namely, the coupling should be separable, i.e., there should be no entanglement between the two systems corresponding to ρ_1, ρ_2. That is, the definition of qRHL used in [17] is Definition 1 with "coupling" replaced by "separable coupling". We will also adopt the separability condition in our definition of expectation-qRHL.[9]

So far, we have basically recapped the definition from [17]. However, that definition only allows us to express Hoare judgments that do not involve expectations since A and B in Definition 1 are subspaces (predicates), not expectations. To define expectation-qRHL, we follow the same idea, but instead of quantifying over only the initial states satisfying the precondition, we quantify over all initial states, and merely require that (the coupling of) the final states satisfies the postexpectation at least as much as (the coupling of the) initial states satisfy the preexpectation. That is:

▶ **Definition 2** (Expectation-qRHL (eqRHL), first attempt). *For expectations A, B, $\{A\}\,\mathfrak{c} \sim \mathfrak{d}\,\{B\}$ holds iff for any ρ_1, ρ_2 with separable coupling ρ, the final states $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$ have a separable coupling ρ' such that $\mathrm{tr}\,\mathsf{A}\rho \leq \mathrm{tr}\,\mathsf{B}\rho'$. (Recall that $\mathrm{tr}\,\mathsf{A}\rho$ indicates how much ρ satisfies A, and analogously $\mathrm{tr}\,\mathsf{B}\rho'$, cf. Section 2.)*

For terminating programs $\mathfrak{c}, \mathfrak{d}$, this definition already works well. Unfortunately, if $\mathfrak{c}, \mathfrak{d}$ do not terminate with probability 1, we have some undesired effects: For example, assume that $\mathfrak{c} = \underline{\mathbf{skip}}$, and $\mathfrak{d}$ is a program that with probability $1-\varepsilon$ does nothing ($\underline{\mathbf{skip}}$), and but with probability ε does not terminate ($\underline{\mathbf{abort}}$). Then $\{\mathsf{A}\}\,\mathfrak{c} \sim \mathfrak{d}\,\{\mathsf{B}\}$ does not hold *for any* A, B. Why? The final states $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$ have trace 1 and $1-\varepsilon$, respectively. Therefore there exists no coupling ρ' of $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$. (It follows from the definition of couplings that the coupling must have the same trace as its marginals.) Hence $\{\mathsf{A}\}\,\mathfrak{c} \sim \mathfrak{d}\,\{\mathsf{B}\}$ does not hold. Similarly, $\{\mathsf{A}\}\,\mathfrak{c} \sim \mathfrak{d}\,\{\mathsf{B}\}$ does not hold whenever there are two input states ρ_1, ρ_2 such that $\mathfrak{c}, \mathfrak{d}$ terminate with different probabilities. (Even if this nontermination only occurs for input states for which A evaluates to 0!) This makes it near impossible to reason about non-terminating programs.[10]

[9] [17] was not able to prove the Frame rule without adding this separability condition. Our reasons for adopting the separability condition are slightly different: we do not have a Frame rule anyway, but for other elementary rules such as the rule Equal in [17] with quantum expectations and quantum variables, it is unclear how to prove them without the separability condition.

[10] Even if we are interested in a Hoare logic with total correctness, this behavior is undesired. Instead, we want that a nontermination with small probability simply introduces some small penalty in the expectations. For example, in the case of non-relational Hoare with total correctness, $\{(1-\varepsilon)\mathsf{id}\}\,\mathsf{A}\,\{\mathsf{id}\}$ means that A is nonterminating with probability $\leq \varepsilon$.

There are a number of approaches how one can circumvent this problem. E.g., one could allow ρ' to be a "subcoupling" instead of a coupling (i.e., its marginals do not have to equal $[\![\mathfrak{c}]\!](\rho_1), [\![\mathfrak{d}]\!](\rho_2)$ but only lower bound them);[11] the subcoupling always exists, even if the traces are not equal. However, we find that adding such "hacks" to the definition makes it more difficult to understand what the definition exactly does in case of non-terminating programs.

Instead, we choose an approach that makes non-termination explicit. That is, when a program does not terminate, we assign a specific state $|\bot\rangle$ to its output, and we allow expectation to explicitly refer to it (e.g., an expectation could assign value 1 to nontermination, and value 0 to termination). The denotation $[\![\mathfrak{c}]\!]^\bot$ defined on page 7 does exactly that. And expectations that live on a space that has an explicit nontermination-state $|\bot\rangle$ were introduced as $\bot$-expectations on page 5. This leads to the following definition:

▶ **Definition 3** (Expectation-qRHL, informal). *For $\bot$-expectations A, B, $\{A\}\,\mathfrak{c} \sim \mathfrak{d}\,\{B\}$ holds iff for any ρ_1, ρ_2 with separable coupling ρ, the final states $[\![\mathfrak{c}]\!]^\bot(\rho_1), [\![\mathfrak{d}]\!]^\bot(\rho_2)$ have a separable coupling ρ' such that $\operatorname{tr} \mathsf{A}\rho \le \operatorname{tr} \mathsf{B}\rho'$.*

Note that a coupling of $[\![\mathfrak{c}]\!]^\bot(\rho_1), [\![\mathfrak{d}]\!]^\bot(\rho_2)$ always exists since $[\![\cdot]\!]^\bot$ is trace-preserving. (Below, we will derive certain specific variants of eqRHL such as eqRHL with total correctness as specific cases of this definition. Also, we will see that subcouling-based definitions can be recovered as special cases in Lemma 9.) By plugging in the definition of couplings into Definition 3, we get the following precise definition:

▶ **Definition 4** (Expectation-qRHL, generic). *Let A, B be $\bot$-expectations and $\mathfrak{c}$, $\mathfrak{d}$ programs. Then $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{gen}}{\approx} \mathfrak{d}\,\{\mathsf{B}\}$ holds iff for any separable mixed $\bot$-bimemory ρ over $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$, there is a separable mixed $\bot$-bimemory ρ' over $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$ such that*

- $\operatorname{tr}_2 \rho' = [\![\mathfrak{c}]\!]^\bot(\operatorname{tr}_2 \rho)$.
- $\operatorname{tr}_1 \rho' = [\![\mathfrak{d}]\!]^\bot(\operatorname{tr}_1 \rho)$.
- $\operatorname{tr} \mathsf{A}\rho \le \operatorname{tr} \mathsf{B}\rho'$.

In this definition, $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$ are isomorphic copies of the set $\mathbf{X}^{\mathbf{all}}$ of variables. That is, while strictly speaking, $[\![\mathfrak{c}]\!]^\bot$ maps mixed $\bot$-memories over $\mathbf{X}^{\mathbf{all}}$ to mixed $\bot$-memories over $\mathbf{X}^{\mathbf{all}}$, we can also see it as mapping mixed $\bot$-memories over $\mathbf{X}_1^{\mathbf{all}}$ to mixed $\bot$-memories over $\mathbf{X}_1^{\mathbf{all}}$. Analogously for $\mathfrak{d}$ and $\mathbf{X}_2^{\mathbf{all}}$. We make use of this in the preceding definition when we apply $[\![\mathfrak{c}]\!]^\bot, [\![\mathfrak{d}]\!]^\bot$ to ρ_1, ρ_2, respectively. In the rest of the paper, we simply call a mixed state (ρ_1, ρ_2)-*coupling* if it is separable and has marginals ρ_1 and ρ_2.

Note that we defined $\bot$-expectations to be invariant under $\mathcal{E}_\bot \otimes \mathcal{E}_\bot$ (page 5), i.e., that they implicitly measure first whether the quantum memories are $|\bot\rangle$. Otherwise, we would not even have $\{\mathsf{A}\}\,\underline{\mathbf{skip}} \overset{\text{gen}}{\approx} \underline{\mathbf{skip}}\,\{\mathsf{A}\}$ (rule Skip below), for example if $\mathsf{A} := \mathsf{proj}(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|\bot\rangle)$. This is because even the program $\underline{\mathbf{skip}}$ measures whether the memory is $|\bot\rangle$ (by definition of $[\![\cdot]\!]^\bot$), so it may change the state if the memory is in a superposition between $|\bot\rangle$ and something else.

Partial/total correctness. The generic definition of eqRHL (Definition 4) allows us to explicitly express in our pre-/postexpectations how nontermination should be treated. While this allows for maximal generality, in practice it might be cumbersome to always have to specify explicitly how the expectations behave on $|\bot\rangle$. Instead, we define below three special cases of eqRHL that hardcode the treatment of $|\bot\rangle$.

[11] This is explored further in Section 4 for special cases of our definition.

The simplest case is eqRHL with total correctness: Here, nontermination is "forbidden", i.e., we assign value 0 to it. Recall from page 4 that for an expectation A, $\mathsf{A} \oplus 0_{\bot\bot}$ is the corresponding $\bot$-expectation. It assigns 0 to a state that has $|\bot\rangle$ in the left or right memory. Hence, eqRHL with total correctness simply means that all pre/postconditions are of the form $\mathsf{A} \oplus 0_{\bot\bot}$. The following definition specifies convenient syntax for this special case:

▶ **Definition 5** (Expectation-qRHL, total)**.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *iff* $\{\mathsf{A} \oplus 0_{\bot\bot}\}\,\mathfrak{c} \overset{\text{gen}}{\sim} \mathfrak{d}\,\{\mathsf{B} \oplus 0_{\bot\bot}\}$.

A second common variant of Hoare logic is "partial correctness". Here we allow nontermination, i.e., if a program does not terminate, we assign the value 1. That is, we use pre/postexpectations of the form $(\mathsf{A} \oplus 0_{\bot\bot}) + T$ where T assigns value 1 when the left or right memory is in state $|\bot\rangle$:

▶ **Definition 6** (Expectation-qRHL, partial)**.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *iff* $\{(\mathsf{A} \oplus 0_{\bot\bot}) + T\}\,\mathfrak{c} \overset{\text{gen}}{\sim} \mathfrak{d}\,\{(\mathsf{B} \oplus 0_{\bot\bot}) + T\}$ *where* $T := \big(\mathsf{proj}(|\bot_1\rangle) \otimes P_{\not\bot}\big) + \big(P_{\not\bot} \otimes \mathsf{proj}(|\bot_2\rangle)\big) + \mathsf{proj}(|\bot_1\rangle \otimes |\bot_2\rangle)$.

Unfortunately, this definition does not necessarily behave as we would like. E.g., if both $\mathfrak{c}$ and $\mathfrak{d}$ terminate with probability $\leq \frac{1}{2}$ on all inputs, then $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ holds for any $\mathsf{A} \leq \mathsf{id}$, B. That is, any property holds with probability $\frac{1}{2}$ for those programs which is not what we would expect! Why does this happen? Since $[\![\mathfrak{c}]\!]^{\bot}(\rho_1), [\![\mathfrak{d}]\!]^{\bot}(\rho_2)$ are 50% nontermination, we can "match up" the nonterminating part of $[\![\mathfrak{c}]\!]^{\bot}(\rho_1)$ with the terminating part of $[\![\mathfrak{d}]\!]^{\bot}(\rho_2)$ and vice versa in the coupling ρ' of the output states. Then $\operatorname{tr} T\rho' = 1$ and thus $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ holds. The problem here is that we treat nontermination as a "wildcard" that is allowed to match any behavior of the other program. While there may be valid use cases for such a notation, we believe that in most cases this is not what we want.

Instead, we define a notion we call "semipartial". In this eqRHL-variant, we allow nontermination, but only when it occurs in the two programs "in sync". I.e., we consider pre/postexpectations that assign 1 to $|\bot\rangle \otimes |\bot\rangle$, but 0 to a state where one program has nonterminated and the other has terminated. Formally:

▶ **Definition 7** (Expectation-qRHL, semipartial)**.** *Let* A, B *be expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{semi}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *iff*

$$\{(\mathsf{A} \oplus 0_{\bot\bot}) + \mathsf{proj}(|\bot_1\rangle \otimes |\bot_2\rangle)\}\,\mathfrak{c} \overset{\text{gen}}{\sim} \mathfrak{d}\,\{(\mathsf{B} \oplus 0_{\bot\bot}) + \mathsf{proj}(|\bot_1\rangle \otimes |\bot_2\rangle)\}.$$

Pure initial states. In many cases, it is much easier to work with the definition of eqRHL correctness if one can assume that the initial states of $\mathfrak{c}, \mathfrak{d}$ are pure states, and that the initial coupling is the tensor product of those states. (No nontrivial correlations.) The following lemma shows that we can do so without loss of generality:

▶ **Lemma 8.** *Let* A, B *be* $\bot$*-expectations and* $\mathfrak{c}$, $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{gen}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *holds iff*

- *for all unit quantum memories* ψ_1, ψ_2 *over* $\mathbf{X}_1$, $\mathbf{X}_2$, *respectively, there is a separable* $\bot$*-mixed bimemory* ρ' *over* $\mathbf{X}_1, \mathbf{X}_2$ *such that*
  - $\operatorname{tr}_2 \rho' = [\![\mathfrak{c}]\!]^{\bot}(\mathsf{proj}(\psi_1))$.
  - $\operatorname{tr}_1 \rho' = [\![\mathfrak{d}]\!]^{\bot}(\mathsf{proj}(\psi_2))$.
  - $\operatorname{tr} \mathsf{A}\,\mathsf{proj}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho'$.[12]

[12] Or equivalently, $\|\sqrt{\mathsf{A}}(\psi_1 \otimes \psi_2)\| \leq \operatorname{tr} \mathsf{B}\rho'$. Or $(\psi_1 \otimes \psi_2)^* \mathsf{A}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho'$.

- *for all unit quantum memories* ψ_1 *over* $\mathbf{X}_1$*, we have* $\operatorname{tr} \mathsf{A} \operatorname{proj}(\psi_1 \otimes |\bot_2\rangle) \leq \operatorname{tr} \mathsf{B}\big([\![\mathfrak{c}]\!]^\bot(\operatorname{proj}(\psi_1)) \otimes \operatorname{proj}(|\bot_2\rangle)\big)$.
- *for all unit quantum memories* ψ_2 *over* $\mathbf{X}_2$*, we have* $\operatorname{tr} \mathsf{A} \operatorname{proj}(|\bot_1\rangle \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\big(\operatorname{proj}(|\bot_1\rangle) \otimes [\![\mathfrak{d}]\!]^\bot(\operatorname{proj}(\psi_2))\big)$.
- $\operatorname{tr} \mathsf{A} \operatorname{proj}(|\bot_1\rangle \otimes |\bot_2\rangle) \leq \operatorname{tr} \mathsf{B} \operatorname{proj}(|\bot_1\rangle \otimes |\bot_2\rangle)$.

Equivalent reformulations. As discussed after Definition 2, an alternative means of trying to circumvent the problem that Definition 2 does handle nonterminating programs well is to use subcouplings instead of couplings.

Here, we show that the notions of eqRHL with partial and total correctness can be equivalently restated in terms of subcouplings (instead of the extended semantics $[\![\cdot]\!]^\bot$ over $\bot$-memories). However, we do not know such an equivalent reformulation for semipartial correctness.

▶ **Lemma 9.** *Let* A*,* B *be expectations and* $\mathfrak{c}$*,* $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *iff for any separable mixed bimemory* ρ *over* $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$*, there is a separable mixed bimemory* ρ' *over* $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$ *such that*

- $\operatorname{tr}_2 \rho' \leq [\![\mathfrak{c}]\!](\operatorname{tr}_2 \rho)$.
- $\operatorname{tr}_1 \rho' \leq [\![\mathfrak{d}]\!](\operatorname{tr}_1 \rho)$.
- $\operatorname{tr} \mathsf{A}\rho \leq \operatorname{tr} \mathsf{B}\rho'$.

▶ **Lemma 10.** *Let* A*,* B *be expectations and* $\mathfrak{c}$*,* $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *holds iff for any separable mixed bimemory* ρ *over* $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$*, there is a separable mixed bimemory* ρ' *over* $\mathbf{X}_1^{\mathbf{all}}, \mathbf{X}_2^{\mathbf{all}}$ *such that*

- $\operatorname{tr}_2 \rho' \leq [\![\mathfrak{c}]\!](\operatorname{tr}_2 \rho)$.
- $\operatorname{tr}_1 \rho' \leq [\![\mathfrak{d}]\!](\operatorname{tr}_1 \rho)$.
- $\operatorname{tr} \rho' \geq \operatorname{tr}[\![\mathfrak{c}]\!](\operatorname{tr}_2 \rho) + \operatorname{tr}[\![\mathfrak{d}]\!](\operatorname{tr}_1 \rho) - \operatorname{tr} \rho$.
- $\operatorname{tr} \mathsf{A}\rho \leq \operatorname{tr} \mathsf{B}\rho' + \operatorname{tr} \rho - \operatorname{tr} \rho'$.

In this definition, $\operatorname{tr} \rho - \operatorname{tr} \rho' \geq 0$ is describe the nonterminating probability. Lemma 9 and Lemma 10 mean that total and partial correctness can be alternatively defined using the concept of subcouplings, without considering the $\bot$-extension of the expectations and programs.

▶ **Lemma 11.** *Let* A*,* B *be expectations and* $\mathfrak{c}$*,* $\mathfrak{d}$ *programs. Then* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$ *(resp.* $\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{par}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$*) holds iff for all unit quantum memories* ψ_1*,* ψ_2 *over* $\mathbf{X}_1^{\mathbf{all}}$*,* $\mathbf{X}_2^{\mathbf{all}}$*, respectively, there is a separable mixed bimemory* ρ *over* $\mathbf{X}_1\mathbf{X}_2$ *such that*

- $\operatorname{tr}_2 \rho \leq [\![\mathfrak{c}]\!](\operatorname{proj}(\psi_1))$.
- $\operatorname{tr}_1 \rho \leq [\![\mathfrak{d}]\!](\operatorname{proj}(\psi_2))$.
- $\operatorname{tr} \mathsf{A}\operatorname{proj}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho$.[13]

  (resp. $\operatorname{tr} \mathsf{A}\operatorname{proj}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho + 1 - \operatorname{tr} \rho$ *and* $1 + \operatorname{tr} \rho \geq \operatorname{tr}[\![\mathfrak{c}]\!](\operatorname{proj}(\psi_1)) + \operatorname{tr}[\![\mathfrak{d}]\!](\operatorname{proj}(\psi_2))$*.)*

5 Description of the rules

We describe the rules of our logic one by one here. Recall that we essentially have four different logics: partial, semipartial, total, and the general case from which the former three are derived. To keep things readable, we only describe the rules for the partial, semipartial, and total case here. (Listed in Figure 1.) In the full version, we state and prove the rules

[13] Or equivalently, $\|\sqrt{\mathsf{A}}(\psi_1 \otimes \psi_2)\| \leq \operatorname{tr} \mathsf{B}\rho$. Or $(\psi_1 \otimes \psi_2)^* \mathsf{A}(\psi_1 \otimes \psi_2) \leq \operatorname{tr} \mathsf{B}\rho$.

SKIP
$$\{\mathsf{A}\}\,\underline{\mathbf{skip}} \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\{\mathsf{A}\}$$

APPLY1
$$\big\{(U\,\mathbf{on}\,\mathbf{X}_1)^*\mathsf{A}(U\,\mathbf{on}\,\mathbf{X}_1)\big\}\,\underline{\mathbf{apply}}\ U\ \underline{\mathbf{to}}\ \mathbf{X} \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\{\mathsf{A}\}$$

EXFALSO
$$\{0\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}$$

INIT1
$$\big\{\mathsf{id}_{\mathbf{X}_1}\otimes(\psi^*\otimes\mathsf{id}_{\neg\mathbf{X}_1})\mathsf{A}(\psi\otimes\mathsf{id}_{\neg\mathbf{X}_1})\big\}\,\mathbf{X}\leftarrow\psi \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\{\mathsf{A}\}$$

SEQ
$$\frac{\{\mathsf{A}\}\,\mathfrak{c}_1 \overset{\text{any}}{\simeq} \mathfrak{d}_1\,\{\mathsf{B}\}\qquad \{\mathsf{B}\}\,\mathfrak{c}_2 \overset{\text{any}}{\simeq} \mathfrak{d}_2\,\{\mathsf{C}\}}{\{\mathsf{A}\}\,\mathfrak{c}_1;\mathfrak{c}_2 \overset{\text{any}}{\simeq} \mathfrak{d}_1;\mathfrak{d}_2\,\{\mathsf{C}\}}$$

CONSEQ
$$\frac{\mathsf{A}'\le\mathsf{A}\qquad \{\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}\qquad \mathsf{B}\le\mathsf{B}'}{\{\mathsf{A}'\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}'\}}$$

SYM
$$\frac{\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}}{\{\mathsf{SWAP}^*\cdot\mathsf{A}\cdot\mathsf{SWAP}\}\,\mathfrak{d} \overset{\text{any}}{\simeq} \mathfrak{c}\,\{\mathsf{SWAP}^*\cdot\mathsf{B}\cdot\mathsf{SWAP}\}}$$

SCALE
$$\frac{\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}\qquad \lambda\in[0,1]}{\{\lambda\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\lambda\mathsf{B}\}}$$

IF1
$$\frac{\{\mathsf{A}_T\}\,\mathfrak{c}_T \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}\qquad \{\mathsf{A}_F\}\,\mathfrak{c}_F \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}}{\{\downarrow^*_{\mathsf{true}}(\mathsf{A}_T)+\downarrow^*_{\mathsf{false}}(\mathsf{A}_F)\}\,\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}_T\ \underline{\mathbf{else}}\ \mathfrak{c}_F \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\mathsf{B}\}}$$

JOINTIF9
$$\frac{\{\mathsf{A}_{t,u}\}\,\mathfrak{c}_t \overset{\text{any}}{\simeq} \mathfrak{d}_u\,\{\mathsf{B}\}\text{ for }t,u\in\{\mathsf{true},\mathsf{false}\}}{\big\{\textstyle\sum_{t,u\in\{\mathsf{true},\mathsf{false}\}}\downarrow^*_{t,u}(\mathsf{A}_{t,u})\big\}\,\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}_{\mathsf{true}}\ \underline{\mathbf{else}}\ \mathfrak{c}_{\mathsf{false}} \overset{\text{any}}{\simeq} \underline{\mathbf{if}}\ N[\mathbf{Y}]\ \underline{\mathbf{then}}\ \mathfrak{d}_{\mathsf{true}}\ \underline{\mathbf{else}}\ \mathfrak{d}_{\mathsf{false}}\,\{\mathsf{B}\}}$$

JOINTIF
$$\frac{\{\mathsf{A}_{\mathsf{true}}\}\,\mathfrak{c}_{\mathsf{true}} \overset{\text{any}}{\simeq} \mathfrak{d}_{\mathsf{true}}\,\{\mathsf{B}\}\qquad \{\mathsf{A}_{\mathsf{false}}\}\,\mathfrak{c}_{\mathsf{false}} \overset{\text{any}}{\simeq} \mathfrak{d}_{\mathsf{false}}\,\{\mathsf{B}\}}{\big\{\downarrow^*_{\mathsf{true},\mathsf{true}}(\mathsf{A}_{\mathsf{true}})+\downarrow^*_{\mathsf{false},\mathsf{false}}(\mathsf{A}_{\mathsf{false}})\big\}\,\underline{\mathbf{if}}\ M[\mathbf{X}]\ \underline{\mathbf{then}}\ \mathfrak{c}_{\mathsf{true}}\ \underline{\mathbf{else}}\ \mathfrak{c}_{\mathsf{false}} \overset{\text{any}}{\simeq} \underline{\mathbf{if}}\ N[\mathbf{Y}]\ \underline{\mathbf{then}}\ \mathfrak{d}_{\mathsf{true}}\ \underline{\mathbf{else}}\ \mathfrak{d}_{\mathsf{false}}\,\{\mathsf{B}\}}$$

WHILE1
$$\frac{\big\{\mathsf{A}\big\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\big\{\downarrow^*_{\mathsf{true}}(\mathsf{A})+\downarrow^*_{\mathsf{false}}(\mathsf{B})\big\}\qquad \underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c}\text{ is terminating}}{\big\{\downarrow^*_{\mathsf{true}}(\mathsf{A})+\downarrow^*_{\mathsf{false}}(\mathsf{B})\big\}\,\underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c} \overset{\text{any}}{\simeq} \underline{\mathbf{skip}}\,\{\mathsf{B}\}}$$

JOINTWHILE
$$\frac{\begin{array}{c}\{\mathsf{A}\}\,\mathfrak{c} \overset{\text{any}}{\simeq} \mathfrak{d}\,\{\downarrow^*_{\mathsf{true},\mathsf{true}}(\mathsf{A})+\downarrow^*_{\mathsf{false},\mathsf{false}}(\mathsf{B})\}\\ \underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c}\text{ or }\underline{\mathbf{while}}\ N[\mathbf{Y}]\ \underline{\mathbf{do}}\ \mathfrak{d}\text{ is terminating}\end{array}}{\{\downarrow^*_{\mathsf{true},\mathsf{true}}(\mathsf{A})+\downarrow^*_{\mathsf{false},\mathsf{false}}(\mathsf{B})\}\,\underline{\mathbf{while}}\ M[\mathbf{X}]\ \underline{\mathbf{do}}\ \mathfrak{c} \overset{\text{any}}{\simeq} \underline{\mathbf{while}}\ N[\mathbf{Y}]\ \underline{\mathbf{do}}\ \mathfrak{d}\,\{\mathsf{B}\}}$$

Figure 1 Rules for total/semipartial/partial eqRHL. In these rules, "any" can be any of "tot", "semi", "par". For any = par, the termination condition in WHILE1 can be replaced by $\mathsf{A}\le\mathsf{id}$, and for any = par, semi the termination condition in JOINTWHILE can be replaced by $\mathsf{A}\le\mathsf{id}$. We refer to the symmetric rules of APPLY1, INIT1, IF1, and WHILE1 (obtained by applying SYM) as APPLY2, INIT2, IF2, and WHILE2.

in the general case. The rules in Figure 1 are then simple consequences of the rules in the general case. The sole exception are rules related to while-loops: here not all of the partial, semipartial, total case follow directly from the general while rule. Those cases that do not follow are proved separately.

5.1 Structural rules

First, we mention the "structural" rules, i.e., rules that do not related to a specific language construct. There is Sym for exchanging the two programs. (In this rule, $\mathsf{SWAP} : \ell^2[\mathbf{X}_2^{\mathbf{all}}] \otimes \ell^2[\mathbf{X}_1^{\mathbf{all}}] \mapsto \ell^2[\mathbf{X}_1^{\mathbf{all}}] \otimes \ell^2[\mathbf{X}_2^{\mathbf{all}}]$ is the unitary operator $\mathsf{SWAP}_\perp : (\psi \otimes \phi) = \phi \otimes \psi.$) ExFalso allows us to show anything from an impossible preexpectation. Seq allows us to analyze the sequential composition of programs. Conseq allows us to weaken a judgment. (The preexpectation can be replaced by a smaller preexpectation, and the postexpectation can be replaced by a larger preexpectation. $\leq$ is the Loewner order). And finally, Scale allows us to scale pre- and postexpectations by a scalar factor.

5.2 One-sided rules

Conceptually simplest are the one-sided rules, i.e., rules that have **skip** on the right (or left) hand side. By combining them with Seq, we can prove facts about pairs of programs one statement at the time. Here, we only describe the rules with **skip** on the right side, the other case is analogous.

Apply: First, consider the Apply1 rule. It is stated (like all our rules), in a backward reasoning style, i.e., for any postexpectation A, the tells us the corresponding preexpectation, here $(U \mathbf{on} \mathbf{X}_1)^* \mathsf{A} (U \mathbf{on} \mathbf{X}_1)$. (Recall that $U \mathbf{on} \mathbf{X}_1$ denotes U applied to $\mathbf{X}_1$.) This is quite intuitive: the left program applies $U \mathbf{on} \mathbf{X}_1$, so the preexpectation corresponding to the postexpectation A is what we get if we apply $U \mathbf{on} \mathbf{X}_1$ and then compute the preexpectation, i.e., $(U \mathbf{on} \mathbf{X}_1)^* \mathsf{A} (U \mathbf{on} \mathbf{X}_1)$. (And it is $(U \mathbf{on} \mathbf{X}_1)^* \mathsf{A} (U \mathbf{on} \mathbf{X}_1)$ and not $\mathsf{A}(U \mathbf{on} \mathbf{X}_1)$ because the latter is not Hermitian and thus not a valid expectation.)

A toy example how to apply this rule: we want to what $\mathbf{x}$ has to be so that it will be $|0\rangle$ after applying a Hadamard H. Thus our postexpectation is $\mathsf{proj}(|0\rangle) \mathbf{on} \mathbf{x}_1$. Applying rule Apply1, we get that $\{\mathsf{B}\}\, \underline{\mathbf{apply}}\ H\ \underline{\mathbf{to}}\ \mathbf{x} \overset{\text{any}}{\sim} \underline{\mathbf{skip}}\, \{\mathsf{proj}(|0\rangle) \mathbf{on} \mathbf{x}_1\}$ for $\mathsf{B} := (H \mathbf{on} \mathbf{x}_1)^* (\mathsf{proj}(|0\rangle) \mathbf{on} \mathbf{x}_1)(H \mathbf{on} \mathbf{x}_1)$. (Here $\overset{\text{any}}{\sim}$ can be any of $\overset{\text{tot}}{\sim}$, $\overset{\text{semi}}{\sim}$, $\overset{\text{par}}{\sim}$.) A simple calculation reveals: $\mathsf{B} = \big(H^* \mathsf{proj}(|0\rangle) H\big) \mathbf{on} \mathbf{x}_1 = \mathsf{proj}(|+\rangle) \mathbf{on} \mathbf{x}_1$. Thus we learned (unsurprisingly) that to get $|0\rangle$, we need to start out with $|+\rangle$.

Init: The rule Init1 rule stated in a similar backwards reasoning way as Apply1, but the preexpectation is somewhat less intuitive. We will illustrate it with a toy example. Assume we want to know what the probability is to measure $|0\rangle$ after initializing a variable $\mathbf{x}$ with $|+\rangle$. That is, our postexpectation is $\mathsf{A} := \mathsf{proj}(|0\rangle) \mathbf{on} \mathbf{x}_1$ and our left program is $\mathbf{x} \leftarrow |+\rangle$. We ask for a suitable preexpectation B in $\{\mathsf{B}\}\, \mathbf{x} \leftarrow |+\rangle \overset{\text{any}}{\sim} \underline{\mathbf{skip}}\, \{\mathsf{A}\}$. The Init1 rule gives us $\mathsf{B} = \mathsf{id}_{\mathbf{x}_1} \otimes \big(\langle +| \otimes \mathsf{id}_{\neg \mathbf{x}_1}\big) \mathsf{A} \big(|+\rangle \otimes \mathsf{id}_{\neg \mathbf{x}_1}\big)$. (Here, $\neg \mathbf{X}_1 := \mathbf{X}_1^{\mathbf{all}} \mathbf{X}_2^{\mathbf{all}} \setminus \mathbf{X}_1$.) By definition of A, we have that $\big(\langle +| \otimes \mathsf{id}_{\neg \mathbf{x}_1}\big) \mathsf{A} \big(|+\rangle \otimes \mathsf{id}_{\neg \mathbf{x}_1}\big) = \langle +|\mathsf{proj}(|0\rangle)|+\rangle \otimes \mathsf{id}_{\neg \mathbf{x}_1} = \frac{1}{2} \mathsf{id}_{\neg \mathbf{x}_1}$. Note that this is not an expectation in our sense because it is an operator on all variable *but* $\mathbf{x}_1$. But by tensoring with $\mathsf{id}_{\mathbf{x}_1}$, we get the expectation $\mathsf{B} = \frac{1}{2}\mathsf{id}$. Thus the preexpectation is $\frac{1}{2}\mathsf{id}$ which intuitively means that, no matter what the initial state, the probability of measuring $|0\rangle$ will be $\frac{1}{2}$, as we would expect.

If: The rule rule If1 allows us to prove a judgment about an if-statement from judgments about the then- and the else-branch. If the preexpectations from the then- and else-branch are A_T and A_F, then the preexpectations for the if-statement is $\downarrow^*_{\mathsf{true}}(\mathsf{A}_T) + \downarrow^*_{\mathsf{false}}(\mathsf{A}_F)$. (Here

$\downarrow_t^*(A) := (M_t \,\mathbf{on}\, \mathbf{X}_1)^* A (M_t \,\mathbf{on}\, \mathbf{X}_1)$.) This is natural since $\downarrow^*_{\mathsf{true}}(\mathsf{A}_T)$ is A_T restricted to the case where the conditional holds, and $\downarrow^*_{\mathsf{false}}(\mathsf{A}_F)$ is A_F restricted to the case where the conditional does not hold.

A toy example: We want to show $\{\mathsf{id}\}\ \underline{\mathbf{if}}\ M[\mathbf{x}]\ \underline{\mathbf{then}}\ \underline{\mathbf{apply}}\ X\ \underline{\mathbf{to}}\ \mathbf{x}\ \underline{\mathbf{else}}\ \underline{\mathbf{skip}} \overset{\text{any}}{\sim} \underline{\mathbf{skip}}\ \{\mathsf{B}\}$ with $\mathsf{B} := \mathsf{proj}(|0\rangle)\,\mathbf{on}\,\mathbf{x}_1$. Here M is a computational basis measurement ($M_{\mathsf{true}} = \mathsf{proj}(|1\rangle)$, $M_{\mathsf{false}} = \mathsf{proj}(|0\rangle)$), and X is the pauli-X matrix (quantum bit flip). That is, with probability 1 (preexpectation is id), if we measure $\mathbf{x}$ in the computational basis, and, in case of outcome 1 flip it, we get $|0\rangle$ (postexpectation B). We derive easily (using rules Apply1 and Skip) that $\{\mathsf{proj}(|1\rangle)\,\mathbf{on}\,\mathbf{x}_1\}\ \underline{\mathbf{apply}}\ X\ \underline{\mathbf{to}}\ \mathbf{x} \overset{\text{any}}{\sim} \underline{\mathbf{skip}}\ \{\mathsf{B}\}$ and $\{\mathsf{proj}(|0\rangle)\,\mathbf{on}\,\mathbf{x}_1\}\ \underline{\mathbf{skip}} \overset{\text{any}}{\sim} \underline{\mathbf{skip}}\ \{\mathsf{B}\}$. From the If1 rule, we then get $\{\mathsf{A}\}\ \underline{\mathbf{if}}\ M[\mathbf{x}]\ \underline{\mathbf{then}}\ \underline{\mathbf{apply}}\ X\ \underline{\mathbf{to}}\ \mathbf{x}\ \underline{\mathbf{else}}\ \underline{\mathbf{skip}} \overset{\text{any}}{\sim} \underline{\mathbf{skip}}\ \{\mathsf{B}\}$ with $\mathsf{A} = \downarrow^*_{\mathsf{true}}(\mathsf{proj}(|1\rangle)\,\mathbf{on}\,\mathbf{x}_1) + \downarrow^*_{\mathsf{false}}(\mathsf{proj}(|0\rangle)\,\mathbf{on}\,\mathbf{x}_1)$. Thus $\mathsf{A} = \mathsf{proj}(|1\rangle)\,\mathbf{on}\,\mathbf{x}_1 + \mathsf{proj}(|0\rangle)\,\mathbf{on}\,\mathbf{x}_1 = \mathsf{id}$, as desired.

While: The While1 rule is similar to the If1 rule. (The preexpectation in the conclusion has the same form.) The main difference is that we need to guess the invariant A because the postexpectation in the premise contains A. The rule also requires us to prove first that the loop is terminating (except in the case of partial correctness). Since this is a statement about a single program (non-relational), it can be shown using existing approaches (e.g., [14]) and is outside the scope of this paper.

5.3 Two-sided rules

The one-sided rules discussed in the previous section allow us to analyze two programs one statement at a time. However, they are not sufficient if want to analyze the relationship of two programs that go in lockstop. (E.g., two while loops that always take the same decision whether to terminate.) For handling if- and while-statements that are in sync, our logic provides the two-sided rules JointIf9, JointIf, and JointWhile. Notice that there are not two-sided analogues to Apply1 and Init1. This is because the resulting rule would be no different from using the one-sided rule twice. However, when random choices happen, two-sided rules are useful. In our case, this happens in if- and while-statements because the measurement of the loop-condition introduces randomness.

If: The JointIf9 rule allows us to compute the preexpectation of two if-statements. It is analogous to the If1 rule, except that the resulting preexpectation is of the form $\sum_{t,u\in\{\mathsf{true},\mathsf{false}\}} \downarrow^*_{t,u}(\mathsf{A}_{t,u})$. (Here $\downarrow^*_{t,u}(\mathsf{A}) := \big((M_t\,\mathbf{on}\,\mathbf{X}_1)\otimes(N_t\,\mathbf{on}\,\mathbf{Y}_2)\big)^*\mathsf{A}\big((M_t\,\mathbf{on}\,\mathbf{X}_1)\otimes(N_t\,\mathbf{on}\,\mathbf{Y}_2)\big)$.) That is, the preexpectations are restricted to all four combinations of true/false for the two if-conditions. And, consequently, we have a premise for each of those four cases. (The rule is called JointIf9 because, in the general case, there are nine cases, due to explicit treatment of non-terminating cases.) For convenience, we additionally state rule JointIf which considers only the cases where the two if-statements are in sync (true/true or false/false).

While: Similar to JointIf, the JointWhile rule allows us to reason about while loops that are in sync. Like with While1, in contrast to JointIf, we need to guess the invariant A. For an example, see Section 6. One difference with the While1 rule is that While1 requires us to prove termination in the semipartial and total case (not in the partical case), while JointWhile requires us to prove termination only in the total case. (Intuitively, this is because in the semipartial case, termination is not required, it is only required that both programs terminate with the same probability.)

6 Example: Quantum Zeno effect

Motivation. In this section, we study (one specific incarnation of) the quantum Zeno effect as an example of application of our logic. The Zeno effect implies that the following processes have the same effect:

- Start with a qubit in state $|0\rangle$. Apply a continuous rotation (with angular velocity ω) to it. (Thus, after time t, the state will have rotated by angle ωt.)
- Start with a qubit in state $|0\rangle$. Continuously observe the state. Namely, at time t, measure whether the qubit has rotated by angle ωt.

The quantum Zeno effect implies that in both processes, the state evolves in the same way (and that the measurement in the second situation always gives answer "yes"). Notice that this means that the measurements can be used to rotate the state.

In our formalization, we will consider the discrete version of this phenomenon: The rotation is split into n rotations by a small angle, and the continuous measurement consists of n measurements. In the limit $n \to \infty$, both processes yield the same state, but if we consider the situation for a concrete value of n, the result of the processes will be slightly different. (And the difference can be quantified in terms of n.) This makes this example a prime candidate for our logic: We want to compare two processes (hence we need relational Hoare logic), but the processes are not exactly equivalent (hence we cannot use qRHL from [17]) but only close to equivalent (and the "amount of equivalence" can be expressed using expectations).

Formalizing the processes. We now formalize the two processes as programs in our language. Let $n \geq 1$ be an integer.

In the first process, we have a continuous rotation, broken down into n small rotations. For simplicity, we will rotate by the angle $\pi/2$ within n steps, thus each small rotation rotates by angle $\frac{\pi}{2n}$. This is described by the rotation matrix $R := \begin{pmatrix} \cos\frac{\pi}{2n} & -\sin\frac{\pi}{2n} \\ \sin\frac{\pi}{2n} & \cos\frac{\pi}{2n} \end{pmatrix}$. Let $\mathbf{y}$ be a variable of type $\{0,1\}$ (i.e., the qubit that is rotated). In order to apply the rotation n times, we will need a counter $\mathbf{x}$ for the while loop. Let $\mathbf{x}$ be a variable of type $\mathbb{Z}$. We will have a loop that continues while (informally speaking) $\mathbf{x} < n$. This is formalized by the projector $P_{<n}$ onto states $|i\rangle$ with $i < n$. I.e., $P_{<n} := \sum_{-\infty<i<n} \mathsf{proj}(|i\rangle)$. In slight abuse of notation, we also write $P_{<n}$ for the binary measurement with Kraus operators $\{P_{<n}, \mathsf{id} - P_{<n}\}$. Furthermore, we need to increase the counter. For this let INCR be the unitary on $\ell^2(\mathbb{Z})$ with $\mathsf{INCR}|i\rangle \mapsto |i+1\rangle$ Then the program that initializes $\mathbf{y}$ with $|0\rangle$ and then applies the rotation R n times can be written as:

$$\mathbf{c} := \mathbf{x} \leftarrow |0\rangle;\ \mathbf{y} \leftarrow |0\rangle;\ \underline{\textbf{while}}\ P_{<n}[\mathbf{x}]\ \underline{\textbf{do}}\ (\underline{\textbf{apply}}\ \mathsf{INCR}\ \underline{\textbf{to}}\ \mathbf{x};\ \underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}) \tag{1}$$

In the second process, instead of applying R, we measure the state in each iteration of the loop. In the first iteration, we expect the original state $\phi_0 := |0\rangle$, and after the i-th iteration, we expect the state $\phi_i := R\phi_{i-1}$ for $i \geq 1$. This can be done using the program $\underline{\textbf{if}}\ \mathsf{proj}(\phi_i)[\mathbf{y}]\ \underline{\textbf{then}}\ \underline{\textbf{skip}}\ \underline{\textbf{else}}\ \underline{\textbf{skip}}$ where we again write in slight abuse of notation $\mathsf{proj}(\phi_i)$ for the corresponding binary measurement. Since the if-statement first measures $\mathbf{y}$ and then executes one of the $\underline{\textbf{skip}}$-branches, this is effectively just a measurement. We abbreviate this as $\underline{\textbf{if}}\ \mathsf{proj}(\phi_i)[\mathbf{y}]$.

However, we cannot simply write $\underline{\textbf{if}}\ \mathsf{proj}(\phi_i)[\mathbf{y}]$ in our loop body, because i should be the value of $\mathbf{x}$. So we need to define the projector that projects onto ϕ_i when $\mathbf{x} = |i\rangle$. This is done by the following projector on $\ell^2[\mathbf{xy}]$: $P_\phi := \sum_i \mathsf{proj}(|i\rangle \otimes \phi_i)$. Then $\underline{\textbf{if}}\ P_\phi[\mathbf{y}]$ will measure whether $\mathbf{y}$ contains ϕ_i whenever $\mathbf{x}$ contains $|i\rangle$.

Armed with that notation, we can now formulate the second process as a program:

$$\mathfrak{d} := \mathbf{x} \leftarrow |0\rangle;\ \mathbf{y} \leftarrow |0\rangle;\ \underline{\textbf{while}}\ P_{<n}[\mathbf{x}]\ \underline{\textbf{do}}\ (\underline{\textbf{apply}}\ \mathsf{INCR}\ \underline{\textbf{to}}\ \mathbf{x};\ \underline{\textbf{if}}\ P_\phi[\mathbf{xy}]) \tag{2}$$

Equivalence of the programs. We claim that the two processes, i.e., the programs $\mathfrak{c}, \mathfrak{d}$ have approximately the same final state in $\mathbf{y}$. Having the same state can be expressed using the "quantum equality" described in Section 2. Specifically, the postexpectation $\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2$ corresponds to $\mathbf{y}_1$ and $\mathbf{y}_2$ having the same state. For example, $\{\mathsf{id}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\}$ implies that the final state of $\mathfrak{c}$ and $\mathfrak{d}$ is the same (if we trace out all variables except $\mathbf{y}_1, \mathbf{y}_2$).[14] The fact that the final states are approximately equal can be expressed by multiplying the preexpectation with a real number close to 1. Specifically, in our case we claim that

$$\{\varepsilon^n \cdot \mathsf{id}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\} \tag{3}$$

Here $\varepsilon := (\cos\frac{\pi}{2n})^2$. This indeed means that the final states of $\mathfrak{c}$ and $\mathfrak{d}$ are the same asymptotically since $\varepsilon^n = (\cos\frac{\pi}{2n})^{2n} \xrightarrow{n\to\infty} 1$.

Warm up. Before we prove (3), we investigate a simpler case as a warm up. We investigate the special where $n = 3$, and instead of a while-loop, we simply repeat the loop body three times.

$$\begin{aligned}\mathfrak{c}' &:= \mathbf{y} \leftarrow |0\rangle;\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y};\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y};\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}\\ \mathfrak{d}' &:= \mathbf{y} \leftarrow |0\rangle;\underline{\textbf{if}}\ \mathsf{proj}(\phi_1)[\mathbf{y}];\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_2)[\mathbf{y}];\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_3)[\mathbf{y}]\end{aligned} \tag{4}$$

We claim:

$$\{\varepsilon^3 \cdot \mathsf{id}\}\,\mathfrak{c}' \overset{\text{tot}}{\sim} \mathfrak{d}'\,\{\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\} \tag{5}$$

First, we strengthen the postcondition. Let $\mathsf{A}_3 := (\mathsf{proj}(\phi_3 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2)$. (This postcondition is intuitively what we expect to (approximately) hold at the end of the execution. It means that $\mathbf{y}_1$ and $\mathbf{y}_2$ are both in state ϕ_3, the result by rotating three times using R. Since $\phi_3 \otimes \phi_3$ is in the image of the projector EQUAL, it follows that $\mathsf{A}_3 \le (\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2)$. By rule Conseq it is thus sufficient to show $\{\varepsilon^3 \cdot \mathsf{id}\}\,\mathfrak{c}' \overset{\text{tot}}{\sim} \mathfrak{d}'\,\{\mathsf{A}_3\}$. And by rule Seq, we can show that by the following sequence of Hoare judgments for some $\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2$:

$$\left\{\varepsilon^3 \cdot \mathsf{id}\right\} \underset{\text{tot}}{\sim} \left\{\mathsf{A}_0\right\} \underset{\text{tot}}{\sim} \left\{\mathsf{A}_1\right\} \underset{\text{tot}}{\sim} \left\{\mathsf{A}_2\right\} \underset{\text{tot}}{\sim} \left\{\mathsf{A}_3\right\} \tag{6}$$

$$\begin{array}{l}\mathbf{y} \leftarrow |0\rangle\\ \mathbf{y} \leftarrow |0\rangle\end{array}\qquad \begin{array}{l}\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}\\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_1)[\mathbf{y}]\end{array}\qquad \begin{array}{l}\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}\\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_2)[\mathbf{y}]\end{array}\qquad \begin{array}{l}\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}\\ \underline{\textbf{if}}\ \mathsf{proj}(\phi_3)[\mathbf{y}]\end{array}$$

(These are four judgments, we just use a more compact notation to put them in one line.) We will derive suitable values $\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2$ by applying our rules backwards from the postcondition.

By applying rule Apply1, we get $\left\{\mathsf{A}_3'\right\}\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y} \overset{\text{tot}}{\sim} \underline{\textbf{skip}}\left\{\mathsf{A}_3\right\}$ where $\mathsf{A}_3' := (R^\dagger\,\mathbf{on}\,\mathbf{y}_1) \circ \mathsf{A}_3$ and where we use $A \circ B$ as an abbreviation for $ABA^\dagger$. And by rule If2 (using rule Skip for its premises), we get

$$\left\{(\mathsf{proj}(\phi_3)\,\mathbf{on}\,\mathbf{y}_2) \circ \mathsf{A}_3' + (1 - \mathsf{proj}(\phi_3)\,\mathbf{on}\,\mathbf{y}_2) \circ \mathsf{A}_3'\right\}\underline{\textbf{skip}} \overset{\text{tot}}{\sim} \underline{\textbf{if}}\ \mathsf{proj}(\phi_3)[\mathbf{y}]\left\{\mathsf{A}_3'\right\}.$$

[14] This is seen as follows: The judgment implies that the finals states are marginals of a state that is invariant under the projector $\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2$, i.e., a state with support in the space $\mathbf{Y}_1 \equiv_{\mathfrak{q}} \mathbf{Y}_2$. That means that this state is invariant under swapping $\mathbf{Y}_1, \mathbf{Y}_2$, and thus the marginals corresponding to $\mathbf{Y}_1$ and $\mathbf{Y}_2$ are equal.

The precondition is lower bounded by $A_2 := (\mathsf{proj}(\phi_3)\,\mathbf{on}\,\mathbf{y}_2) \circ \mathsf{A}'_3$. (The second term corresponds to the measurement failing to measure ϕ_3, in this case all is lost anyway, so we remove that term.) Hence (with rules Seq and Conseq), $\{\mathsf{A}_2\}$ **apply** R **to** $\mathbf{y} \overset{\text{tot}}{\sim}$ **if** $\mathsf{proj}(\phi_3)[\mathbf{y}]\,\{\mathsf{A}_3\}$ as desired in (6).

Analogously, we can instantiate

$$\mathsf{A}_1 := (\mathsf{proj}(\phi_2)\,\mathbf{on}\,\mathbf{y}_2) \circ (R^*\,\mathbf{on}\,\mathbf{y}_1) \circ \mathsf{A}_2 \quad \text{and} \quad \mathsf{A}_0 := (\mathsf{proj}(\phi_1)\,\mathbf{on}\,\mathbf{y}_2) \circ (R^*\,\mathbf{on}\,\mathbf{y}_1) \circ \mathsf{A}_1$$

in (6). We can simplify the expressions for $\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2$ some more. We have

$$\begin{aligned}\mathsf{A}_2 &= (\mathsf{proj}\big(\phi_3\big)\,\mathbf{on}\,\mathbf{y}_2) \circ (R^*\,\mathbf{on}\,\mathbf{y}_1) \circ \big(\mathsf{proj}(\phi_3 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\big) \\ &= \mathsf{proj}\big(R^*\phi_3 \otimes \mathsf{proj}(\phi_3)\phi_3\big)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2 = \mathsf{proj}(\phi_2 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\end{aligned}$$

And

$$\begin{aligned}\mathsf{A}_1 &= (\mathsf{proj}\big(\phi_2\big)\,\mathbf{on}\,\mathbf{y}_2) \circ (R^*\,\mathbf{on}\,\mathbf{y}_1) \circ \big(\mathsf{proj}(\phi_2 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2\big) \\ &= \mathsf{proj}\big(R^*\phi_2 \otimes \mathsf{proj}(\phi_2)\phi_3\big)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2 = \varepsilon\,\mathsf{proj}(\phi_1 \otimes \phi_2)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2.\end{aligned}$$

(Note the slight difference: instead of $\mathsf{proj}(\phi_3)\phi_3$ have $\mathsf{proj}(\phi_2)\phi_3$ here, which simplifies to $\phi_2 \cdot \phi_2^*\phi_3 = \phi_2 \cdot \sqrt{\varepsilon}$.) Analogously

$$\mathsf{A}_0 = \varepsilon^2\,\mathsf{proj}(\phi_0 \otimes \phi_1)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2.$$

It is left to show the first judgment in (6), namely $\{\varepsilon^3 \cdot \mathsf{id}\}\,\mathbf{y} \leftarrow |0\rangle \overset{\text{tot}}{\sim} \mathbf{y} \leftarrow |0\rangle\,\{\mathsf{A}_0\}$. By rules Init1 and Init2 (starting from the right), we have

$$\begin{aligned}\Big\{\varepsilon^3 \cdot \mathsf{id}\Big\} &\overset{(**)}{=} \Big\{\mathsf{id}_{\mathbf{y}_2} \otimes \big(\langle 0|_{\mathbf{y}_2} \otimes \mathsf{id}_{\neg\mathbf{y}_2}\big) \circ \varepsilon^2(\mathsf{proj}(\phi_1)\,\mathbf{on}\,\mathbf{y}_2)\Big\}\ \underline{\mathbf{skip}} \overset{\text{tot}}{\sim} \mathbf{y} \leftarrow |0\rangle \\ &\Big\{\varepsilon^2(\mathsf{proj}(\phi_1)\,\mathbf{on}\,\mathbf{y}_2)\Big\} \overset{(*)}{=} \Big\{\mathsf{id}_{\mathbf{y}_1} \otimes \big(\langle 0|_{\mathbf{y}_1} \otimes \mathsf{id}_{\neg\mathbf{y}_1}\big) \circ \mathsf{A}_0\Big\}\,\mathbf{y} \leftarrow |0\rangle \overset{\text{tot}}{\sim} \underline{\mathbf{skip}}\,\Big\{\mathsf{A}_0\Big\}. \quad (7)\end{aligned}$$

Here $(*)$ uses that $\phi_0 = |0\rangle$ and thus $\langle 0|\mathsf{proj}(\phi_0)|0\rangle = 1$, and $(**)$ uses that $\phi_1^*\phi_0 = \sqrt{\varepsilon}$ and thus $\langle 0|\mathsf{proj}(\phi_1)|0\rangle = \varepsilon$.

The first judgment in (6) then follows by rule Seq.

This completes the analysis, we have shown (5).

Analysis of the while-programs. Given the experiences from the analysis of the special case (the programs from (4)), we now can solve the original problem, namely analyzing the programs $\mathfrak{c}, \mathfrak{d}$ from (1),(2).

As before, we can replace the postcondition in (3) by the stronger postcondition $\mathsf{B} := (\mathsf{proj}(|n\rangle) \otimes |n\rangle \otimes \phi_n \otimes \phi_n)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2)$. By rule Conseq, it is sufficient to show $\{\varepsilon^n \cdot \mathsf{id}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$. By rule Seq, this follows if we can show

$$\Big\{\varepsilon^n \cdot \mathsf{id}\Big\}\ \begin{matrix}\mathbf{x} \leftarrow |0\rangle \\ \overset{\text{tot}}{\sim}\ \mathbf{x} \leftarrow |0\rangle\end{matrix}\ \Big\{\mathsf{D}\Big\}\ \begin{matrix}\mathbf{y} \leftarrow |0\rangle \\ \overset{\text{tot}}{\sim}\ \mathbf{y} \leftarrow |0\rangle\end{matrix}\ \Big\{\mathsf{C}\Big\}\ \begin{matrix}\text{while}_{\mathfrak{c}} \\ \overset{\text{tot}}{\sim}\ \text{while}_{\mathfrak{d}}\end{matrix}\ \Big\{\mathsf{B}\Big\} \quad (8)$$

with

$$\begin{aligned}\text{while}_{\mathfrak{c}} &:= \underline{\mathbf{while}}\ P_{<n}[\mathbf{x}]\ \underline{\mathbf{do}}\ (\underline{\mathbf{apply}}\ \mathsf{INCR}\ \underline{\mathbf{to}}\ \mathbf{x};\ \underline{\mathbf{apply}}\ R\ \underline{\mathbf{to}}\ \mathbf{y}) \\ \text{while}_{\mathfrak{d}} &:= \underline{\mathbf{while}}\ P_{<n}[\mathbf{x}]\ \underline{\mathbf{do}}\ (\underline{\mathbf{apply}}\ \mathsf{INCR}\ \underline{\mathbf{to}}\ \mathbf{x};\ \underline{\mathbf{if}}\ P_\phi[\mathbf{xy}])\end{aligned}$$

for suitably chosen expectations C, D.

To prove the last judgment $\{\mathsf{C}\}\,\text{while}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \text{while}_{\mathfrak{d}}\,\{\mathsf{B}\}$ in (8), we use rule JointWhile. This rule requires us to come up with a loop invariant A. To understand what the right loop invariant is, we draw from our experiences in the special case. There, we had defined the expectations $\mathsf{A}_0, \dots, \mathsf{A}_3$, where A_i described the state of the programs right after the i-th application of $\underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}$ and $\underline{\textbf{if}}\ \mathsf{proj}(\phi_i)[\mathbf{y}]$. We had

$$\mathsf{A}_i = \varepsilon^{2-i}\,\mathsf{proj}(\phi_i \otimes \phi_{i+1})\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2 \text{ for } i = 0,1,2 \qquad \text{and} \qquad \mathsf{A}_3 = \mathsf{proj}(\phi_3 \otimes \phi_3)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2$$

One sees easily that this would generalize as

$$\mathsf{A}_i = \varepsilon^{n-i-1}\,\mathsf{proj}(\phi_i \otimes \phi_{i+1})\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2 \qquad \text{for } i < n$$
$$\text{and} \qquad \mathsf{A}_n = \mathsf{proj}(\phi_n \otimes \phi_n)\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2$$

for values $n \neq 3$. Thus we expect that these expectations A_i also hold in the programs $\text{while}_{\mathfrak{c}}$, $\text{while}_{\mathfrak{d}}$ after the i-th iteration (or before the $(i+1)$-st iteration). Additionally, we keep track of the counter $\mathbf{x}$, which should be $|i\rangle$ after the i-th iteration (or before the $(i+1)$-st iteration). This would be expressed by the expectation $\mathsf{proj}(|i\rangle \otimes |i\rangle)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2$. Thus, for the i-th iteration, we use the "conjunction"

$$\mathsf{A}_i^{\mathbf{x}} := \mathsf{A}_i \cdot (\mathsf{proj}(|i\rangle \otimes |i\rangle)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2)$$
$$= \begin{cases} \varepsilon^{n-i-1}\,\mathsf{proj}(|i\rangle \otimes |i\rangle \otimes \phi_i \otimes \phi_{i+1})\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 & (i < n) \\ \mathsf{proj}(|n\rangle \otimes |n\rangle \otimes \phi_n \otimes \phi_n)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 & (i = n) \end{cases}$$

(Note that $\cdot$ is not generally a sensible operation on expectations. But in this case, $\mathsf{fv}(\mathsf{A}_i) = \mathbf{y}_1\mathbf{y}_2$ and $\mathsf{fv}(\mathsf{proj}(|i\rangle \otimes |i\rangle)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2) = \mathbf{x}_1\mathbf{x}_2$, so the expectations commute and their product is again an expectation.)

The final loop invariant A is then the "disjunction" of the $\mathsf{A}_i^{\mathbf{x}}$ for $i = 0, \dots, n-1$, meaning that in every iteration, one of the A_i should hold. (We do not include $\mathsf{A}_i^{\mathbf{x}}$ with $i = n$ here because when applying the JointWhile rule, we only need the invariant to hold when the loop guard was passed.) We define $\mathsf{A} := \sum_{i=0}^{n} \mathsf{A}_i^{\mathbf{x}}$. (In general, summation is not a sensible operation representation of "disjunction", but in the present case, all summands are orthogonal.)

We have now derived a suitable candidate for the invariant A to use in rule JointWhile. We stress that the above argumentation (involving words like "disjunction" and "conjunction" of expectations, and claims that an expectation "holds" at a certain point) was not a formally well-defined argument, merely an explanation how we arrived at our specific choice for A. From the formal point of view, all we will need in the following are the definitions of $\mathsf{A}, \mathsf{A}_i^{\mathbf{x}}$. The rest of the argument above was semi-formal motivation.

We will now show the rightmost judgment in (8), namely $\{\mathsf{C}\}\,\text{while}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \text{while}_{\mathfrak{d}}\,\{\mathsf{B}\}$ (for some suitable C). If we apply rule JointWhile (with A as defined above) to this, we get the premise[15]

$$\Big\{\mathsf{A}\Big\}\ \overbrace{\underbrace{\underline{\textbf{apply}}\ \mathsf{INCR}\ \underline{\textbf{to}}\ \mathbf{x};\ \underline{\textbf{apply}}\ R\ \underline{\textbf{to}}\ \mathbf{y}}_{\overset{\text{tot}}{\sim}\ \underline{\textbf{apply}}\ \mathsf{INCR}\ \underline{\textbf{to}}\ \mathbf{x};\ \underline{\textbf{if}}\ P_\phi[\mathbf{xy}]}}^{=:\text{body}_{\mathfrak{c}}}\ \Big\{\underbrace{P^{\text{both}}_{<n} \circ \mathsf{A} + (P^{\text{none}}_{<n}) \circ \mathsf{B}}_{=:\mathsf{C}'}\Big\} \tag{9}$$
$$=:\text{body}_{\mathfrak{d}}$$

[15] We also additionally get the premise that $\text{while}_{\mathfrak{c}}$ is terminating. This can be shown with techniques from prior work (e.g., [14]) and is quite obvious in the present case. Alternatively, we could have stated this example with respect to partial correctness instead of total correctness. In that case, we do not need to prove termination.

with $P^{\text{both}}_{<n} := P_{<n} \otimes P_{<n}\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2$ and $P^{\text{none}}_{<n} := (\mathsf{id} - P_{<n}) \otimes (\mathsf{id} - P_{<n})\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2$. (Here we write $A \circ B$ as an abbreviation for $ABA^\dagger$.) By applying rules If2, Apply2, and twice Apply1 (with Seq in between), we get

$$\big\{(\mathsf{INCR}\,\mathbf{on}\,\mathbf{x}_1) \circ (R\,\mathbf{on}\,\mathbf{y}_1) \circ (\mathsf{INCR}\,\mathbf{on}\,\mathbf{x}_2) \circ \mathsf{B}_2\big\}\,\mathrm{body}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \mathrm{body}_{\mathfrak{d}}\,\big\{\mathsf{C}'\big\}$$

where $\mathsf{B}_2 := (P_\phi\,\mathbf{on}\,\mathbf{x}_2\mathbf{y}_2) \circ \mathsf{C}' + (\mathsf{id} - P_\phi\,\mathbf{on}\,\mathbf{x}_2\mathbf{y}_2) \circ \mathsf{C}'$. Since $\mathsf{B}_2 \geq (P_\phi\,\mathbf{on}\,\mathbf{x}_1\mathbf{y}_1) \circ \mathsf{C}'$, by rule Conseq we can weaken this to

$$\big\{\mathsf{A}'\big\}\,\mathrm{body}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \mathrm{body}_{\mathfrak{d}}\,\big\{\mathsf{C}'\big\}$$
$$\text{with} \qquad \mathsf{A}' := \underbrace{(\mathsf{INCR}\,\mathbf{on}\,\mathbf{x}_1) \circ (R\,\mathbf{on}\,\mathbf{y}_1) \circ (\mathsf{INCR}\,\mathbf{on}\,\mathbf{x}_2) \circ (P_\phi\,\mathbf{on}\,\mathbf{x}_2\mathbf{y}_2)}_{=:L} \circ \mathsf{C}'$$

If we can show that $\mathsf{A} \leq \mathsf{A}'$ then we have proven (9). By definition of $\mathsf{A}^{\mathbf{x}}_i$, L, R, P_ϕ, INCR, $P^{\text{both}}_{<n}$, we have

$$\begin{aligned} L \circ P^{\text{both}}_{<n} \circ \mathsf{A}^{\mathbf{x}}_i &= \varepsilon^{n-i-1}\,\mathsf{proj}(\mathsf{INCR}^*|i\rangle \otimes \mathsf{INCR}^*|i\rangle \otimes R^*\phi_i \otimes \mathsf{proj}(\phi_i)\phi_{i+1})\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 \\ &= \varepsilon^{n-i}\,\mathsf{proj}(|i-1\rangle \otimes |i-1\rangle \otimes \phi_{i-1} \otimes \phi_i)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 = \mathsf{A}^{\mathbf{x}}_{i-1}. \end{aligned}$$

And $L \circ P^{\text{both}}_{<n} \circ \mathsf{A}^{\mathbf{x}}_n = 0$ since $P^{\text{both}}_{<n} \circ \mathsf{A}^{\mathbf{x}}_n = 0$. Thus $L \circ P^{\text{both}}_{<n} \circ \mathsf{A} = \sum_{i=0}^{n-1} A^{\mathbf{x}}_{i-1} \geq \sum_{i=0}^{n-2} A^{\mathbf{x}}_i$. And by definition of B, L, R, P_ϕ, INCR, $P^{\text{none}}_{<n}$, we have

$$\begin{aligned} L \circ P^{\text{none}}_{<n} \circ \mathsf{B} &= \mathsf{proj}\big(\mathsf{INCR}^*|n\rangle \otimes \mathsf{INCR}^*|n\rangle \otimes R^*\phi_n \otimes \mathsf{proj}(\phi_n)\phi_{n+1}\big)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 \\ &= \mathsf{proj}\big(|n-1\rangle \otimes |n-1\rangle \otimes \phi_{n-1} \otimes \phi_n\big)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2\mathbf{y}_1\mathbf{y}_2 = \mathsf{A}^{\mathbf{x}}_{n-1}. \end{aligned}$$

Thus $\mathsf{A}' = L \circ \mathsf{C}' \geq \sum_{i=0}^{n-2} A^{\mathbf{x}}_i + A^{\mathbf{x}}_{n-1} = \mathsf{A}$. Thus we have proven (9). By rule JointWhile, this implies $\{\mathsf{C}'\}\,\mathrm{while}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \mathrm{while}_{\mathfrak{d}}\,\{\mathsf{B}\}$ with C' as defined in (9). With $\mathsf{C} := A^{\mathbf{x}}_0 \leq \mathsf{C}'$, $\{\mathsf{C}\}\,\mathrm{while}_{\mathfrak{c}} \overset{\text{tot}}{\sim} \mathrm{while}_{\mathfrak{d}}\,\{\mathsf{B}\}$ follows by rule Conseq. This is the rightmost judgment in (8).

Using rules Init1, Init2, and Seq, we get $\{\mathsf{D}\}\,\mathbf{y} \leftarrow |0\rangle \overset{\text{tot}}{\sim} \mathbf{y} \leftarrow |0\rangle\,\{\mathsf{C}\}$ with $\mathsf{D} := \varepsilon^n \cdot (\mathsf{proj}(|0\rangle \otimes |0\rangle)\,\mathbf{on}\,\mathbf{x}_1\mathbf{x}_2)$. (This is done very similarly to (7).) This shows the middle judgment in (8).

Also using rules Init1, Init2, and Seq, we get $\{\varepsilon^n \cdot \mathsf{id}\}\,\mathbf{x} \leftarrow |0\rangle \overset{\text{tot}}{\sim} \mathbf{x} \leftarrow |0\rangle\,\{\mathsf{D}\}$. This shows the leftmost judgment in (8).

Thus we have shown the three judgments in (8). By rule Seq, it follows that $\{\varepsilon^n \cdot \mathsf{id}\}\,\mathfrak{c} \overset{\text{tot}}{\sim} \mathfrak{d}\,\{\mathsf{B}\}$. Since $\mathsf{B} \leq (\mathsf{EQUAL}\,\mathbf{on}\,\mathbf{y}_1\mathbf{y}_2)$, by rule Conseq, we get (3).

References

1. Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In *FOSAD 2012/2013 Tutorial Lectures*, pages 146–166. Springer, 2014. doi:10.1007/978-3-319-10082-1_6.
2. Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Crypto 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, 2011.
3. Gilles Barthe, Benjamin Grégoire, Federico Olmedo, and Santiago Zanella Béguelin. CertiCrypt: Computer-aided cryptographic proofs in Coq. http://certicrypt.gforge.inria.fr/. Accessed 2018-10-24.
4. Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL 2009*, pages 90–101. ACM, 2009. doi:10.1145/1480881.1480894.

5 Gilles Barthe, Justin Hsu, Mingsheng Ying, Nengkun Yu, and Li Zhou. Relational proofs for quantum programs. *Proc. ACM Program. Lang.*, 4:21:1–21:29, 2019. Proceedings of POPL 2020. Full version is arXiv:1901.05184v2. doi:10.1145/3371089.

6 Nick Benton. Simple relational correctness proofs for static analyses and program transformations. In *POPL '04*, pages 14–25. ACM, 2004. doi:10.1145/964001.964003.

7 Rohit Chadha, Paulo Mateus, and Amílcar Sernadas. Reasoning about imperative quantum programs. *ENTCS*, 158:19–39, 2006. doi:10.1016/j.entcs.2006.04.003.

8 John B. Conway. *A course in functional analysis*. Number 96 in Graduate texts in mathematics. Springer, 2nd ed edition, 1997.

9 John B. Conway. *A course in operator theory*. Number 21 in Graduate studies in mathematics. American Mathematical Society, Providence, RI, 2000.

10 Ellie D'Hondt and Prakash Panangaden. Quantum weakest preconditions. *Mathematical. Structures in Comp. Sci.*, 16(3):429–451, 2006. doi:10.1017/S0960129506005251.

11 Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. Proof rules for the correctness of quantum programs. *Theoretical Computer Science*, 386(1):151–166, 2007. doi:10.1016/j.tcs.2007.06.011.

12 Yoshihiko Kakutani. A logic for formal verification of quantum programs. In Anupam Datta, editor, *ASIAN 2009*, pages 79–93, Berlin, Heidelberg, 2009. Springer.

13 Dexter Kozen. A probabilistic PDL. In *STOC '83*, pages 291–297, New York, NY, USA, 1983. ACM. doi:10.1145/800061.808758.

14 Yangjia Li and Mingsheng Ying. Algorithmic analysis of termination problems for quantum programs. *Proc. ACM Program. Lang.*, 2:35:1–35:29, 2018. Proceedings of POPL 2019. doi:10.1145/3158123.

15 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 10th anniversary edition, 2010.

16 Dominique Unruh. dominique-unruh/qrhl-tool: Prototype proof assistant for qRHL. GitHub, 2018. URL: https://github.com/dominique-unruh/qrhl-tool.

17 Dominique Unruh. Quantum relational Hoare logic. *Proc. ACM Program. Lang.*, 3:33:1–33:31, January 2019. Proceedings of POPL 2019. Full version is arXiv:1802.03188 [quant-ph]. doi:10.1145/3290346.

18 Mingsheng Ying. Floyd–Hoare logic for quantum programs. *ACM Trans. Program. Lang. Syst.*, 33(6):19:1–19:49, 2012. doi:10.1145/2049706.2049708.