Quantum Abstract Interpretation

Nengkun Yu
CQSI, FEIT
University of Technology Sydney
Australia
nengkunyu@gmail.com

Jens Palsberg*
Computer Science Department
University of California, Los Angeles (UCLA)
California, USA
palsberg@ucla.edu

Abstract

In quantum computing, the basic unit of information is a qubit. Simulation of a general quantum program takes exponential time in the number of qubits, which makes simulation infeasible beyond 50 qubits on current supercomputers. So, for the understanding of larger programs, we turn to static techniques. In this paper, we present an abstract interpretation of quantum programs and we use it to automatically verify assertions in polynomial time. Our key insight is to let an abstract state be a tuple of projections. For such domains, we present abstraction and concretization functions that form a Galois connection and we use them to define abstract operations. Our experiments on a laptop have verified assertions about the Bernstein-Vazirani, GHZ, and Grover benchmarks with 300 qubits.

CCS Concepts: **• Computer systems organization → Quantum computing**; **• Software and its engineering → Formal software verification**.

Keywords: quantum programming, scalability, abstract interpretation

ACM Reference Format:
Nengkun Yu and Jens Palsberg. 2021. Quantum Abstract Interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '21), June 20–25, 2021, Virtual, Canada.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3453483.3454061

1 Introduction

In the 1980s, Feynman [29, 30] introduced the idea of quantum computing, Benioff [9] described a quantum model of Turing machines, and Deutsch [24, 25] showed how quantum gates can function like classical logic gates. Feynman noted that although a classical computer can simulate the behavior of an n-particle system that evolves according to the laws of quantum mechanics, such simulation is inefficient and needs exponential time and space. Feynman's call to action was to regard the particles themselves as a *quantum computer* that appears to be exponentially more efficient.

Driven by a desire to realize Feynman's vision of great computational power, major efforts have been devoted to building quantum computers. In 2020, the world's largest quantum computer has 72 qubits [57], and by 2024, we are likely to have quantum computers with hundreds of qubits [34]. Along with the hardware efforts, researchers have designed quantum programming languages [4, 36, 44, 49, 51, 52] and quantum programming platforms such as Scaffold [1], Quipper [32], QWIRE [45], Silq [14], Microsoft's LIQUi|⟩ [56] and Q# [53], Google's Cirq [54], and IBM's Qiskit [3]. In such languages, researchers have implemented programs for quantum machine learning [13, 18], variational quantum algorithms [42, 47] applied to quantum chemistry [20], and quantum approximate optimization algorithms [5, 27].

> *How can we check that a quantum program satisfies key correctness criteria?*

For classical computing, we have many approaches for checking correctness; do they carry over to quantum computing?

At one end of the spectrum, we have dynamic techniques such as *simulation*. In quantum computing, simulation scales poorly because the simulation of a general quantum program with n qubits requires working with 2^n complex numbers. The exponential blow-up makes simulation infeasible beyond 50 qubits on current supercomputers [58]. Looking ahead to larger quantum computers, we note that 300 qubits mean 2^{300} complex numbers, which is more than the number of atoms in the known universe.

At the other end of the spectrum, we have static techniques. For quantum computing, those include static verification methods that researchers have applied to quantum programs and quantum cryptographic protocols [2, 6–8, 21, 26, 28, 37, 38, 40, 48, 59, 60, 62]. However, those methods have been demonstrated only for programs with few qubits. Recent work by Hietala et al. [35] and by Chareton et al.

*Also with Challenge Institute for Quantum Computation, UCLA.

PLDI '21, June 20–25, 2021, Virtual, Canada

ACM ISBN 978-1-4503-8391-2/21/06...$15.00
https://doi.org/10.1145/3453483.3454061

[22] have used Coq and Why3 to automatically check the proofs of correctness for a variety of quantum programs. The static techniques also include logical methods that researchers have used to develop a dynamic logic [19], a predicate transformer semantics [61], etc. Those logical methods stem from the Birkhoff-von Neumann quantum logic [15] and the observation that the projections in a Hilbert space form an orthomodular lattice [39]. The many mathematical properties of projections make them versatile for thinking about the correctness of quantum programs. Recently, researchers have used projections both for static verification [55, 64] and for run-time verification [41]. However, all those methods require exponential space, which limits scalability.

In this paper, we break through the exponential barrier for deriving useful information about quantum programs. Our approach rests on a central idea:

> *Rather than focusing on the whole quantum state, we focus on parts.*

Our notion of a *part* is a well-known and extensively used concept in quantum science: the *reduced density matrix*. Intuitively, the whole quantum state can be represented by a density matrix, while a part of the state can be represented by a reduced density matrix. For example, for a program with 20 qubits, which means that the state can be represented by 2^{20} complex numbers, we might track just 19 small $2^2 \times 2^2$ reduced density matrices that focus on the qubit pairs $\{1,2\}, \{2,3\}, \cdots, \{19,20\}$. For comparison, 2^{20} is about a million, while $19 \times 2^2 \times 2^2 = 304$. When the number of qubits grows beyond fifty, tracking the whole state becomes infeasible, while tracking reduced density matrices stays tractable.

Here is an analogy with static analysis of integer variables in classical computing. The full density matrix is like a polyhedron that approximates the values of all the program variables, whereas a tuple of reduced density matrices is like a tuple of polyhedra, each over a subset of those program variables.

Our key insight is that we can approximate each reduced density matrix by a projection, which we call an approximately reduced density matrix. This enables us to define an abstract state to be a tuple of projections. Now we need a notion of state transition between such abstract states, so we bring in abstract interpretation [23], which so far has been done mainly for classical computing. Perdrix [46] presented an abstract interpretation of quantum programs that is sound but lacks a Galois connection between the concrete and abstract domains.

In this paper, we present a new abstract interpretation of quantum programs. For our notion of abstract states, we present abstraction and concretization functions that form a Galois connection and we use them to define abstract operations. Each abstract step first concretizes to a more fine-grained abstract domain, then does an abstract operation on that domain, and finally abstracts back to the original abstract domain. We avoid concretizing all the way to the concrete domain where we would need exponential space. Similar to classical abstract interpretation we have that abstracting from R to T followed by abstracting from T to S can be different from abstracting from R to S. Our example for showing this difference uses quantum entanglement. In the other direction, we have an equality. Specifically, concretizing from S to T followed by concretizing from T to R is the same as concretizing from S to R.

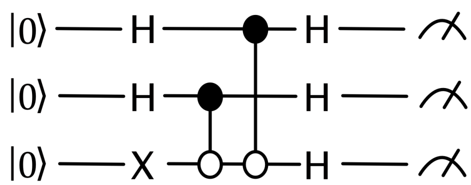
As an example of how our abstract interpretation can be useful, we use it to automatically verify assertions in polynomial time. For example, a key assertion about Grover's search algorithm [33] is that the quantum state is in the span of two vectors that both are expressed as tensor products. We can specify that assertion and then use abstract interpretation to check it. First, we run the abstract interpretation, which produces an abstraction of the state of the quantum program. Second, we abstract the assertion to the same format as the abstract states, and third, we check that the abstract state satisfies the abstracted assertion. Our correctness theorem shows that if the check succeeds, then the assertion is correct. Other quantum algorithms have similar correctness criteria, including the amplitude amplification algorithm [16, 17], which plays a central role in achieving exponential speedup over classical algorithms [11, 12].

We have implemented our approach in Java. Our experiments on a laptop have verified assertions about the Bernstein-Vazirani algorithm [10], GHZ circuit, and Grover algorithm [33] benchmarks with 300 qubits. This shows that our approach scales well and handles programs that are out of reach for simulation.

The rest of the paper. First, we introduce our running example (Section 2), recall quantum computing concepts (Section 3), and recall the notion of a projection (Section 4). Then we define abstract states (Section 5) and abstract operations (Section 6), and we show how to check assertions (Section 7) and how to prove the correctness of assertion checking (Section 8). Finally, we present our experimental results (Section 9) and conclude (Section 10). Most of our proofs are in the supplementary material. Our implementation and our benchmarks are available in the ACM Digital Library.

2 Example

Our running example is the GHZ program for 3 qubits, see Figure 1. We will walk through the example and state the challenge of the paper specifically for the example. Along the way, we recall key concepts of quantum computing.



circuit: 3 qubits	// Dirac notation & vector notation: // $\|000\rangle$ // = $(1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T$
H(0)	// $\frac{1}{\sqrt{2}}(\|000\rangle + \|001\rangle)$ // = $\frac{1}{\sqrt{2}}(1\ 1\ 0\ 0\ 0\ 0\ 0\ 0)^T$
H(1)	// $\frac{1}{2}(\|000\rangle + \|001\rangle + \|010\rangle + \|011\rangle)$ // = $\frac{1}{2}(1\ 1\ 1\ 1\ 0\ 0\ 0\ 0)^T$
X(2)	// $\frac{1}{2}(\|100\rangle + \|101\rangle + \|110\rangle + \|111\rangle)$ // = $\frac{1}{2}(0\ 0\ 0\ 0\ 1\ 1\ 1\ 1)^T$
CNOT(1,2)	// $\frac{1}{2}(\|010\rangle + \|011\rangle + \|100\rangle + \|101\rangle)$ // = $\frac{1}{2}(0\ 0\ 1\ 1\ 1\ 1\ 0\ 0)^T$
CNOT(0,2)	// $\frac{1}{2}(\|001\rangle + \|010\rangle + \|100\rangle + \|111\rangle)$ // = $\frac{1}{2}(0\ 1\ 1\ 0\ 1\ 0\ 0\ 1)^T$
H(0)	// $\frac{1}{\sqrt{8}}(\|000\rangle - \|001\rangle + \|010\rangle + \|011\rangle$ // $+ \|100\rangle + \|101\rangle + \|110\rangle - \|111\rangle)$ // = $\frac{1}{\sqrt{8}}(1\ {-1}\ 1\ 1\ 1\ 1\ 1\ {-1})^T$
H(1)	// $\frac{1}{2}(\|000\rangle - \|011\rangle + \|100\rangle + \|111\rangle)$ // = $\frac{1}{2}(1\ 0\ 0\ {-1}\ 1\ 0\ 0\ 1)^T$
H(2)	// $\frac{1}{\sqrt{2}}(\|000\rangle - \|111\rangle)$ // = $\frac{1}{\sqrt{2}}(1\ 0\ 0\ 0\ 0\ 0\ 0\ {-1})^T$
assert state in span	// the assertion is true $\{\ \|000\rangle,\quad \|111\rangle\ \}$
measure 0..2.	

Figure 1. The GHZ program for 3 qubits.

By the way, the final state in Figure 1 can be computed in different ways and with fewer gates. We chose the example because it helps us illustrate some points.

Circuit for GHZ. The quantum diagram at the top of Figure 1 shows the GHZ program for 3 qubits. A qubit is a vector of length two that contains two complex values.

The GHZ program creates a superposition of two states. In the first state, all qubits are 0, while in the second state, all qubits are 1. In the parlance of quantum computing, the superposition is a maximally entangled state.

In the diagram, each of the three horizontal rows depicts a qubit register that holds a single qubit. Time goes from left to right and the initial value of each qubit register is the qubit $|0\rangle$, as shown on the left. The qubit $|0\rangle$ is written in Dirac notation; it can also be written as the column vector:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

After initialization, the first operations on the qubit registers are H, H, and X, which are these specific unitary matrices:

$$\mathrm{H} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \qquad \mathrm{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

In quantum computing, the analogs of logic gates are quantum *gates*, which mathematically are represented by matrices. Hence, we will use the two terms interchangeably.

Next up is a 2-qubit operation that is depicted as a filled circle connected to an open circle. This operation is known as CNOT and is the following matrix:

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Right after the first CNOT is a second CNOT, followed by three H gates. Finally, to the far right of the diagram, we see depictions of three meters that specify that at this point, a measurement of each qubit will happen.

Syntax of GHZ. In addition to the diagram notation, a quantum program can also be represented as text, like we normally represent a program. In Figure 1, the text on the left shows the GHZ program for 3 qubits. The first line says that the program uses 3 qubits, after which we have a linear ordering of the eight gates in the program. Notice that while the circuit diagram suggests that some operations can take place in parallel, the textual notation makes the parallelism less obvious. In the text, each qubit register has a number; the first row of the diagram corresponds to register 0, the second row corresponds to register 1, and the third row corresponds to register 2. After the eight lines with the operations from the diagram comes an assert statement that says that the final state is in the span of the two vectors $|000\rangle$ and $|111\rangle$. Finally, the last line says that we end the program by measuring all three qubits 0..2, similar to what the diagram shows on the far right.

The states of the computation. The states of the computation are shown in Figure 1 in the comments to the right, both in Dirac notation (in the middle column) and in standard vector notation (just below). The initial state is $|000\rangle$, where qubit register 0 is to the right, qubit register 1 is in the middle, and qubit register 2 is to the left. Notice the "endianness": the qubit registers 0, 1, 2 are ordered top to bottom in the figure, but right to left in the Dirac notation. The state $|000\rangle$ can also be represented as a vector of length 8, namely $(1\,0\,0\,0\,0\,0\,0\,0)^T$, where T denotes transposition. Every state vector has length 1, as always in quantum computing.

By the way, we would love to display $(1\,0\,0\,0\,0\,0\,0\,0)^T$ as a column vector but this would take a lot of vertical space. So, when we display a row vector and then transpose it, we are merely trying to show a column vector while saving space.

The assertion. The final state of the computation, before measurement, is $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$. Thus, the stated assertion is correct: we can express the final state as a linear combination of $|000\rangle$ and $|111\rangle$, so the final state is in the span of $|000\rangle$ and $|111\rangle$.

The measurement will produce either the bit string 000 or the bit string 111, each bitstring with probability $\frac{1}{2}$.

The challenge. We can easily simulate the program in Figure 1 and print the states along the way, including the final state. Thus, simulation lets us verify the assertion for the GHZ program for 3 qubits. However, let us turn our attention to a version of GHZ for 300 qubits, which is a program with 899 gates. Instead of working with state vectors of length $2^3 = 8$, now we are working with state vectors of length 2^{300}, which is larger than the number of atoms in the known universe (as we are fond of mentioning). Thus, simulation of GHZ for 300 qubits is infeasible. However, the GHZ for 300 qubits has an assertion that looks much like the assertion for the case of 3 qubits: the final state is in the span of $|00\ldots0\rangle$ and $|11\ldots1\rangle$. The challenge is: how can we automatically verify this assertion in polynomial time? More generally, how can we do this for other quantum programs and their assertions?

Our approach. We will do an abstract interpretation of the program. Specifically, we will work with abstract states, rather than concrete states, and we will execute abstract operations, rather than the concrete gates in the program. Each of our abstract states is of polynomial size, and each of our abstract operations takes polynomial time. This enables us to do abstract interpretation in polynomial time. In the remainder of the paper, we will go into details of how this works. However, first, we will cover some background material on quantum computing and on linear algebra, to make the paper self-contained as much as possible.

3 Background: Quantum Computing

This section presents the background and notations of quantum information and quantum computation, mainly according to the textbook by Nielsen and Chuang [43].

3.1 Preliminaries

We use the notation $[n] = \{0, 1, \cdots, n-1\}$, the notation $\setminus$ to denote set minus. We use $|s|$ to denote the cardinality of set s.

In this paper, we focus on finite-dimensional vector space $\mathbb{C}^d$ of complex vectors. Linear *operators* are linear mappings between such vector spaces. Operators between d-dimensional vector spaces are represented by $d \times d$ matrices, denoted by $\mathbb{C}^{d\times d}$. I is used to denote the identity matrix. The Hermitian conjugate of an operator A is denoted by $A^\dagger = (A^T)^*$, where A^T is the transpose of A, and B^* is the complex conjugate of B. An operator A is *Hermitian* if $A = A^\dagger$. A Hermitian operator A is positive semi-definite if A has non-negative eigenvalues only. The trace of a matrix A is the sum of the entries on the main diagonal, that is, $\mathrm{Tr}(A) = \sum_i A_{ii}$. An operator U is *unitary* if its Hermitian conjugate is its own inverse, that is, $U^\dagger U = UU^\dagger = I$.

We assume that the reader is familiar with Dirac notation and with the linear-algebra concepts of tensor product, orthonormal basis, inner product and outer product of vectors, and Hilbert spaces. We use Dirac notation, $|\psi\rangle$, to denote a complex column vector in $\mathbb{C}^d$. The inner product of two vectors $|\psi\rangle$ and $|\phi\rangle$ is denoted by $\langle\psi|\phi\rangle \in \mathbb{C}$, which is the usual matrix product of the Hermitian conjugate of $|\psi\rangle$, denoted by $\langle\psi|$, and vector $|\phi\rangle$. The outer product of two vectors $|\psi\rangle$ and $|\phi\rangle$ is denoted by $|\psi\rangle\langle\phi| \in \mathbb{C}^{d\times d}$, which is the usual matrix product of the vector $|\psi\rangle$, and the Hermitian conjugate of $|\phi\rangle$, denoted by $\langle\phi|$. The Euclidean norm of a vector $|\psi\rangle$ is denoted by $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle}$.

3.2 Quantum States

The state space of a *qubit*, or quantum bit, is a 2-dimensional Hilbert space $\mathbb{C}^2$. One important orthonormal basis of a qubit system is the *computational* basis with $|0\rangle = (1,0)^T$ and $|1\rangle = (0,1)^T$. Another important basis, called the $\pm$ basis, consists of $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Thus, $\mathrm{H}|0\rangle = |+\rangle$ and $\mathrm{H}|1\rangle = |-\rangle$.

The state space of multiple qubits is the tensor product of single-qubit state spaces. For example, the classical bitstring 00 can be encoded by $|0\rangle \otimes |0\rangle$ (written $|0\rangle|0\rangle$ or even $|00\rangle$ for short) in the Hilbert space $\mathbb{C}^2 \otimes \mathbb{C}^2$. The Hilbert space for an n-qubit system is $(\mathbb{C}^2)^{\otimes n} \cong \mathbb{C}^{2^n}$.

A quantum state $|\psi\rangle$ is a unit vector, that is, $\||\psi\rangle\| = 1$. We can also represent $|\psi\rangle$ by the density matrix $\rho = |\psi\rangle\langle\psi|$. Every density matrix ρ is positive semi-definite and satisfies $\mathrm{Tr}(\rho) = 1$. Our quantum states are so-called *pure* states; thus our use of density matrices is limited to pure states.

3.3 Quantum Programs

In a quantum program, each instruction is a unitary matrix, such as H, X, CNOT that we discussed in Section 2, or a Toffoli gate (also CCNOT gate), which is the following 8×8

matrix:

$$\text{CCNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

People often refer to quantum programs as quantum circuits, and to unitary matrices as gates. A quantum program p consists of an instruction sequence $U_{F_1} \cdots U_{F_{|p|}}$ and it operates on an n qubit register. The notation U_{F_l} means that the gate operates on the qubits in the list F_l. The initial state is $|0^n\rangle$ and the meaning of the program is the matrix product

$$U_{F_{|p|}} \cdots U_{F_1} |0^n\rangle$$

where we need to interpret U_{F_l} appropriately. The idea is that U_{F_l} applies to the register(s) in F_l while leaving the rest of the registers untouched. Specifically, we can regard U_{F_l} as a unitary matrix that applies to the entire n-qubit register in the following way. For example, F_l has size 1 and $F_l = (i)$, then we can view U_{F_l} as the following unitary matrix that applies to an n-qubit register:

$$(\otimes_{k>i} I) \otimes U \otimes (\otimes_{0 \leq j<i} I)$$

where I denotes the one-qubit identity matrix. Notice that the formula uses U without subscript, which is reasonable because the formula applies U to a specific qubit.

For compactness, we will use the following shorter notation,

$$(\otimes_{k>i} I) \otimes U \otimes (\otimes_{0 \leq j<i} I) \quad = \quad U \otimes I_{[n]\setminus\{i\}},$$

where $I_{[n]\setminus\{i\}}$ is the identity matrix on qubits $[n] \setminus \{i\}$. When we use this notation, we will always give the identity matrices subscripts that will denote the qubits they are associated with.

For a different perspective on the above definition, let us rewrite it using a different notation. For any $k_0, k_1, \cdots, k_{n-1} \in \{0, 1\}$,

$$U_{F_l}(|k_{n-1} \cdots k_0\rangle) \quad = \quad |k_{n-1} \cdots k_{i+1}\rangle \, (U|k_i\rangle) \, |k_{i-1} \cdots k_0\rangle,$$

where the tensor products are implicit.

Similarly, a two-qubit unitary matrix U_{F_l} applied to registers $F_l = (i, j)$ can be regarded as the following unitary matrix applied to an n-qubit register

$$U_{F_l} \otimes I_{[n]\setminus\{i,j\}}.$$

We say that this achieves an *expansion* of a unitary matrix via a tensor product. Notice that the formula uses U_{F_l} with its subscript so that we know the order in which U accepts qubits i and j.

We will sometimes leave the expansion implicit and simply say that a unitary operator U describes a computation step from $|\psi\rangle$ to $U|\psi\rangle$. Similarly, if we represent the quantum state as a density matrix ρ and leave the expansion implicit, then a computation step with U is $U\rho U^\dagger$.

3.4 Reduced Density Matrices

For the purpose of defining quantum abstract interpretation, we will use the standard quantum-science concept of a *reduced density matrix*.

Let $\mathbb{C}^{d_1}, \mathbb{C}^{d_2}$ be the Hilbert spaces of two quantum systems considered in isolation. Then the composite system has a state space modeled by the tensor product $\mathbb{C}^{d_1} \otimes \mathbb{C}^{d_2}$. The notion of *partial trace* is needed to extract the state of a subsystem. Formally, the partial trace over $\mathbb{C}^{d_1}$ is a mapping $\mathrm{Tr}_1(\cdot)$ from operators on $\mathbb{C}^{d_1} \otimes \mathbb{C}^{d_2}$ to operators on $\mathbb{C}^{d_2}$ defined by the following equation: $\mathrm{Tr}_1(|\varphi_1\rangle\langle\psi_1| \otimes |\varphi_2\rangle\langle\psi_2|) = \langle\psi_1|\varphi_1\rangle \cdot |\varphi_2\rangle\langle\psi_2|$ for all $|\varphi_1\rangle, |\psi_1\rangle \in \mathbb{C}^{d_1}$ and $|\varphi_2\rangle, |\psi_2\rangle \in \mathbb{C}^{d_2}$ together with linearity. The partial trace $\mathrm{Tr}_2(\cdot)$ over $\mathbb{C}^{d_2}$ can be defined symmetrically. Suppose that we have a composite system of two subsystems with state spaces $\mathbb{C}^{d_1}, \mathbb{C}^{d_2}$, respectively, and it is in density state ρ. Then the states of the first and second subsystems can be described by $\mathrm{Tr}_2(\rho), \mathrm{Tr}_1(\rho)$, respectively.

For example, if the subsystems are both single qubits, and they are maximally entangled; that is, in state $|\Phi\rangle = \frac{1}{\sqrt{2}}|00\rangle + |11\rangle)$ or equivalently

$$|\Phi\rangle\langle\Phi| = \frac{1}{2}(|00\rangle\langle 00| + |00\rangle\langle 11| + |11\rangle\langle 00| + |11\rangle\langle 11|)$$

then the partial traces $\mathrm{Tr}_1(|\Phi\rangle\langle\Phi|) = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$ and $\mathrm{Tr}_2(|\Phi\rangle\langle\Phi|) = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$ describe states of the second and first subsystems, respectively.

Notice the loss of precision when viewing the entangled system $|\Phi\rangle$ as two subsystems—these two traces also would arise from a system that was not entangled, and all four states 00, 01, 10, and 11 were equally likely, that is

$$\rho = \frac{|00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 10| + |11\rangle\langle 11|}{4}.$$

Thus, from those two traces, we cannot recover whether the entire state is entangled.

The notion of partial trace can be generalized to n-qubit system: for an n-qubit density matrix ρ and any $T \subseteq [n]$, such as $\{1\}$ or $\{2, 3\}$ (containing systems 2 and 3), the joint state of subsystems T is

$$\rho_T = \mathrm{Tr}_{[n]\setminus T}[\rho],$$

where the complementary system $[n] \setminus T$ is traced out. Moreover, the notion of partial trace can be directly generalized to a general n-qubit operator. It is widely known that the partial trace preserves the positive semi-definiteness [43].

4 Background: Projections

Our development of quantum abstract interpretation centers around the standard mathematical concept of orthogonal *projection*. Since the early days of quantum mechanics, researchers have used projections to describe key phenomena in quantum mechanics. In particular, the seminal paper *The Logic of Quantum Mechanics* [15] used orthogonal projections as atomic propositions of a quantum logic. An orthogonal projection matrix P satisfies

$$P = P^\dagger = P^2.$$

Their definition is a bit more restrictive than the classical definition that a matrix P is a projection if $P = P^2$. Their more restrictive definition makes sense because it is a good fit for defining quantum logic, as they showed. Following their lead, we will use orthogonal projections. For simplicity, we will just call them projections throughout this paper.

For example, the following matrix is an orthogonal projection, as one can check easily.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The above matrix has rank 1 and it projects any vector of length 4 onto a space of dimension 1.

Each projection P is in a one-to-one correspondence with a linear subspace $S_P = \{v \mid Pv = v\}$. Throughout this paper, we do not distinguish a projection and the corresponding subspace; this simplifies our presentation significantly. The correspondence between projections and subspaces enables a natural partial order of the set of projections. Specifically, for projections P, Q, we say that

$$P \subseteq Q \text{ iff } S_P \subseteq S_Q$$

For two projections P, Q of the same dimensions, we note that $P \subseteq Q$ iff $QP = P$.

Every projection is positive semi-definite. For any a positive semi-definite matrix A, its *support*, written $\text{supp}(A)$, is the subspace spanned by the eigenvectors of A with nonzero eigenvalues. Birkhoff and Von Neumann [15] defined that for a density matrix ρ and a projection P,

$$\rho \text{ satisfies } P \text{ iff } \text{supp}(\rho) \subseteq P.$$

One can show easily that ρ satisfies P iff $P\rho = P\rho P = \rho$.

Our development of quantum abstract interpretation uses three operations on projections:

- intersection of projections (written $\cap$),
- traceout of a projection followed by finding the support (written $\text{supp}(\text{Tr}_s P)$, where we trace out set s of projection P), and
- expansion of a projection via tensor product (written $P \otimes I_s$, where we expand projection P based on set s).

The following lemmas describe relationships among those three operations. We will use those lemmas to prove the correctness of quantum abstract interpretation in an "algebraic" style. In the statements of the lemmas, we assume $s, s_i, s_j \subseteq [n]$, that P, P_1, P_2 and Q, Q_1, Q_2 are n-qubit projections, that A is a positive semi-definite matrix of an n-qubit system, and ρ is a quantum state.

Lemma 4.1. *If projections $P_1 \subseteq Q_1$ and $P_2 \subseteq Q_2$, then $(P_1 \cap P_2) \subseteq (Q_1 \cap Q_2)$. In particular, $P \subseteq (Q_1 \cap Q_2)$ iff $(P \subseteq Q_1 \wedge P \subseteq Q_2)$.*

Lemma 4.2. $U\text{supp}(A)U^\dagger = \text{supp}(UAU^\dagger)$. *If* $\text{supp}(\rho) \subseteq P$, *then* $\text{supp}(U\rho U^\dagger) \subseteq UPU^\dagger$.

Lemma 4.3. $\text{supp}(\text{Tr}_s(\text{supp}(A))) = \text{supp}(\text{Tr}_s A)$.

Lemma 4.4. *If* $s_i \cap s_j = \emptyset$, *then* $\text{Tr}_{s_j}(\text{Tr}_{s_i} A) = \text{Tr}_{s_i}(\text{Tr}_{s_j} A) = \text{Tr}_{s_i \cup s_j} A$.

Lemma 4.5. *If U applies to qubits $F \subseteq s$, then*

$$U(\text{Tr}_{[n]\setminus s} A)U^\dagger = \text{Tr}_{[n]\setminus s} UAU^\dagger$$

where U is regarded as unitary matrix $U \otimes I_{s \setminus F}$ applying on qubits in s at the left-hand side, and is regarded as unitary matrix $U \otimes I_{[n]\setminus F}$ applying on qubits in $[n]$ at the right-hand side.

Lemma 4.6. *For projection $P_{[n]\setminus s}$ on qubits $[n] \setminus s$,* $\text{supp}(\text{Tr}_s A) \subseteq P_{[n]\setminus s}$ *iff* $\text{supp}(A) \subseteq P_{[n]\setminus s} \otimes I_s$.

The first five lemmas can be proved by the definition of partial trace and support. We prove the last lemma in the supplementary material.

5 Concrete States, Abstract States, and a Galois Connection

We begin our development of quantum abstract interpretation with a general definition of abstract domains. As we will see, the concrete domain can be viewed as a particular abstract domain. We will also define Galois connections between abstract domains. As a special case, this provides a Galois connection between the concrete domain and any abstract domain.

5.1 Definitions

Recall from Section 3 that we can represent a concrete state v (a vector) as a projection $vv^\dagger$ (a matrix), which is known as a *density matrix*. For a program with n qubits, numbered from 0 to $n-1$, a density matrix is a giant projection whose dimension is $2^n \times 2^n$.

How do we abstract a giant projection? Our idea is to let an abstract state be a tuple of small projections. Specifically, for any integer $1 \leq m \leq 2^n$ and m-tuple $S = (s_1, \cdots, s_m)$ with $s_i \subseteq [n]$, we can define an abstract domain $AbsDom(S)$ as follows

$$\begin{aligned} AbsDom(S) \quad = \quad \{ \; & (P_{s_1}, \cdots, P_{s_m}) \mid \\ & P_{s_i} = P_{s_i}^\dagger = P_{s_i}^2 \in \mathbb{C}^{2^{|s_i|} \times 2^{|s_i|}} \; \} \end{aligned}$$

The idea is that for any concrete state P, we can define an abstract state that, for each s_i of interest, contains a matrix P_{s_i} that focuses entirely on the state of the qubit registers in s_i.

In the quantum circuit model, the rank of a concrete state is equal to 1, while the rank of P_{s_i}, an entry of an abstract state, can be greater than 1.

Notice that because S is a tuple, we are allowed to have an s_i in S multiple times, which is no better than having s_i in S a single time. We use a tuple, rather than a set, because it made our implementation straightforward.

We define $[n]^m$ to denote the m-tuple $([n], ..., [n])$, which contains m copies of $[n]$.

Recall that in our setting, a density matrix is a projection of size $2^n \times 2^n$, and the concrete domain is the space of such density matrices. Thus, as a special case of $AbsDom(S)$, we have

$$\text{(concrete domain)}^m \quad = \quad AbsDom([n]^m)$$

where $\text{(concrete domain)}^m$ denotes the set of m-tuples in which each entry is a density matrix.

Notice the vast number of possibilities for defining abstract domains. For each m, there are $(2^n)^m = 2^{nm}$ different abstract domains.

We define an ordering $\sqsubseteq$ on $AbsDom(S)$ as follows. Suppose $S = (s_1, \cdots, s_m)$. If $\mathcal{P}, \mathcal{Q} \in AbsDom(S)$, and

$$\begin{aligned} \mathcal{P} &::= (P_{s_1}, P_{s_2}, \ldots, P_{s_m}) \\ \mathcal{Q} &::= (Q_{s_1}, Q_{s_2}, \ldots, Q_{s_m}) \end{aligned}$$

then we define $\mathcal{P} \sqsubseteq \mathcal{Q}$ iff $\forall i : P_{s_i} \subseteq Q_{s_i}$.

For $S = (s_1, \cdots, s_m)$ and $T = (t_1, \cdots, t_m)$, we define $S \trianglelefteq T$ (pronounced "T is *finer* than S") iff $\forall i : s_i \subseteq t_i$.

For $S = (s_1, \cdots, s_m)$, $T = (t_1, \cdots, t_m)$ and $S \trianglelefteq T$, then we define two mappings $\alpha_{T\to S}$ and $\gamma_{S\to T}$:

$$\begin{aligned} \alpha_{T\to S} &: AbsDom(T) \to AbsDom(S) \\ \alpha_{T\to S}(Q_{t_1}, \cdots, Q_{t_m}) &= (P_{s_1}, \cdots, P_{s_m}) \quad \text{where} \\ P_{s_i} &= \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{Tr}_{t_j \setminus s_i} Q_{t_j}) \end{aligned}$$

$$\begin{aligned} \gamma_{S\to T} &: AbsDom(S) \to AbsDom(T) \\ \gamma_{S\to T}(P_{s_1}, \cdots, P_{s_m}) &= (Q_{t_1}, \cdots, Q_{t_m}) \quad \text{where} \\ Q_{t_j} &= \bigcap_{s_i : s_i \subseteq t_j} P_{s_i} \otimes I_{t_j \setminus s_i} \end{aligned}$$

We say that $\alpha_{T\to S}$ is an abstraction function, and we say that $\gamma_{S\to T}$ is a concretization function. Notice that the definitions of $\alpha_{T\to S}$ and $\gamma_{S\to T}$ are built based on the three operations on projections that we listed in Section 4. Intuitively, $\alpha_{T\to S}$ tends to build a small projection from multiple larger projections, while $\gamma_{S\to T}$ tends to build a large projection from multiple smaller projections.

The abstraction function $\alpha_{T\to S}$ can be interpreted as follows. For each Q_{t_j}, arguably, our best choice of estimating P_{s_i} with $s_i \subseteq t_j$ is $\mathrm{supp}(\mathrm{Tr}_{t_j \setminus s_i} Q_{t_j})$ because this action traces out all information in qubits $t_j \setminus s_i$, and $\mathrm{supp}(\cdot)$ is used for preserving the structure of projections. For a given tuple $(Q_{t_1}, \cdots, Q_{t_m})$, we gather all the information about P_{s_i} from each Q_{t_j} by $\cap$.

The concretization function $\gamma_{S\to T}$ can be interpreted as follows. For each P_{s_i}, arguably, our best choice of estimating Q_{t_j} with $s_i \subseteq t_j$ is $P_{s_i} \otimes I_{t_j \setminus s_i}$. For given $(P_{s_1}, \cdots, P_{s_m})$, we gather all the information about Q_{t_j} from each P_{s_i} by $\cap$.

Note though that in some cases, the dimensions of those projections are the same. In more detail, $\gamma_{S\to T}$ constructs Q_{t_j} by possibly expanding some projections into larger projections and then intersecting them. Dually, $\alpha_{T\to S}$ constructs P_{s_i} by possibly tracing out some qubits from some matrices (and finding the support) and then intersecting them.

For example, let $R = (\{1,2,3\}, \{1,2,4\})$, $T = (\{1,2\}, \{4\})$ and $S = (\{2\}, \{4\})$ with $\mathcal{R} = (R_{1,2,3}, R_{1,2,4})$ such that

$$\begin{aligned} R_{1,2,3} &= (|\Phi\rangle\langle\Phi| + |01\rangle\langle 01|) \otimes |0\rangle\langle 0| \\ R_{1,2,4} &= |000\rangle\langle 000| + |010\rangle\langle 010| \end{aligned}$$

where $|\Phi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$ is a two-qubit maximally entangled state. We compute $\alpha_{R\to T}(\mathcal{R}) = (Q_{1,2}, Q_4)$, where

$$\begin{aligned} Q_{1,2} &= |01\rangle\langle 01|, \\ Q_4 &= |0\rangle\langle 0|, \end{aligned}$$

where the first equality is according to

$$\begin{aligned} &\mathrm{supp}(|\Phi\rangle\langle\Phi| + |01\rangle\langle 01|) \cap \mathrm{supp}(|00\rangle\langle 00| + |01\rangle\langle 01|) \\ = \; &|01\rangle\langle 01|. \end{aligned}$$

Therefore,

$$\alpha_{T\to S} \circ \alpha_{R\to T}(\mathcal{R}) = (|1\rangle\langle 1|, |0\rangle\langle 0|).$$

On the other hand, $\alpha_{R\to S}(\mathcal{R}) = (P_1, P_4)$, where

$$\begin{aligned} P_2 &= \mathrm{supp}(\mathrm{Tr}_1 |\Phi\rangle\langle\Phi| + |1\rangle\langle 1|) \cap \mathrm{supp}(|0\rangle\langle 0| + |1\rangle\langle 1|) \\ &= |0\rangle\langle 0| + |1\rangle\langle 1|, \\ P_4 &= |0\rangle\langle 0|. \end{aligned}$$

Therefore in general

$$\alpha_{T\to S} \circ \alpha_{R\to T}(\mathcal{R}) \neq \alpha_{R\to S}(\mathcal{R}).$$

5.2 Properties

Now we show that $\alpha_{T\to S}$ and $\gamma_{S\to T}$ are monotonic, and that $\alpha_{[n]\to S}$ and $\gamma_{S\to [n]}$ form a Galois connection.

Lemma 5.1 (Abstraction is monotonic)**.** *Suppose $S \trianglelefteq T$. $\forall \mathcal{P}, \mathcal{Q} \in AbsDom(T)$: if $\mathcal{P} \sqsubseteq \mathcal{Q}$, then $\alpha_{T\to S}(\mathcal{P}) \sqsubseteq \alpha_{T\to S}(\mathcal{Q})$.*

Lemma 5.2 (Concretization is monotonic)**.** *Suppose $S \trianglelefteq T$. $\forall \mathcal{P}, \mathcal{Q} \in AbsDom(S)$: if $\mathcal{P} \sqsubseteq \mathcal{Q}$, then $\gamma_{S\to T}(\mathcal{P}) \sqsubseteq \gamma_{S\to T}(\mathcal{Q})$.*

Theorem 5.1 (Weak Galois connection)**.** *Suppose $S \trianglelefteq T$. $\forall \mathcal{P} \in AbsDom(S)$: $\forall \mathcal{Q} \in AbsDom(T)$: if $\mathcal{Q} \sqsubseteq \gamma_{S\to T}(\mathcal{P})$, then $\alpha_{T\to S}(\mathcal{Q}) \sqsubseteq \mathcal{P}$. Moreover, if $\mathcal{Q} = \alpha_{[n]^m\to T}(\mathcal{R})$ for some $\mathcal{R} \in AbsDom([n]^m)$: $\mathcal{Q} \sqsubseteq \gamma_{S\to T}(\mathcal{P})$ iff $\alpha_{T\to S}(\mathcal{Q}) \sqsubseteq \mathcal{P}$.*

Proof. Suppose

$$\begin{aligned}\mathcal{P} &= (P_{s_1}, \cdots, P_{s_m})\\ \mathcal{Q} &= (Q_{t_1}, \cdots, Q_{t_m}).\end{aligned}$$

We have

$$\begin{array}{lrcl} & \mathcal{Q} & \sqsubseteq & \gamma_{S\to T}(\mathcal{P}) \\ \Leftrightarrow & Q_{t_j} & \subseteq & \bigcap_{s_i : s_i \subseteq t_j} P_{s_i} \otimes I_{t_j \setminus s_i} \quad (\forall j) \\ \Leftrightarrow & Q_{t_j} & \subseteq & P_{s_i} \otimes I_{t_j\setminus s_i} \\ & & & \quad (\forall j, s_i : s_i \subseteq t_j) \\ \Leftrightarrow & \mathrm{supp}(\mathrm{Tr}_{t_j\setminus s_i} Q_{t_j}) & \subseteq & P_{s_i} \quad (\forall j, s_i : s_i \subseteq t_j) \\ \Rightarrow & \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{Tr}_{t_j\setminus s_i} Q_{t_j}) & \subseteq & P_{s_i} \quad (\forall i) \\ \Leftrightarrow & \alpha_{T\to S}(\mathcal{Q}) & \sqsubseteq & \mathcal{P} \end{array}$$

In the second step, we use the definition of $\gamma_{S\to T}$; in the third step, we use Lemma 4.1; in the fourth step, we use Lemma 4.6; in the fifth step, we use Lemma 4.1; in the last step, we use the definition of $\alpha_{T\to S}$.

If $\mathcal{Q} = \alpha_{[n]^m \to T}(\mathcal{R})$, we have that $\mathcal{R} = (R, \cdots, R)$ is an m-tuple of a giant projection R:

$$Q_{t_j} = \mathrm{supp}(\mathrm{Tr}_{[n]\setminus t_j} R).$$

From $\mathcal{Q} \sqsubseteq \gamma_{S\to T}(\mathcal{P})$, we use the above argument up to the fourth step to have

$$\begin{array}{lrcl} & \mathrm{supp}(\mathrm{Tr}_{t_j\setminus s_i} Q_{t_j}) & \subseteq & P_{s_i} \ (\forall j, s_i : s_i \subseteq t_j) \\ \Leftrightarrow & \mathrm{supp}(\mathrm{Tr}_{t_j\setminus s_i} \mathrm{supp}(\mathrm{Tr}_{[n]\setminus t_j} R)) & \subseteq & P_{s_i} \ (\forall j, s_i : s_i \subseteq t_j) \\ \Leftrightarrow & \mathrm{supp}(\mathrm{Tr}_{t_j\setminus s_i} \mathrm{Tr}_{[n]\setminus t_j} R) & \subseteq & P_{s_i} (\text{by Lemma 4.3}) \\ \Leftrightarrow & \mathrm{supp}(\mathrm{Tr}_{[n]\setminus s_i} R) & \subseteq & P_{s_i} (\text{by Lemma 4.4}) \\ \Leftrightarrow & \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{Tr}_{[n]\setminus s_i} R) & \subseteq & P_{s_i} (\text{by Lemma 4.1}) \\ \Leftrightarrow & \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{Tr}_{t_j\setminus s_i} Q_{t_j}) & \subseteq & P_{s_i} (\text{by Lemma 4.3}) \\ \Leftrightarrow & \alpha_{T\to S}(\mathcal{Q}) & \sqsubseteq & \mathcal{P}. \end{array}$$

□

Theorem 5.2 (Galois connection)**.** *$\forall \mathcal{P} \in AbsDom(S)$: $\forall \mathcal{Q} \in AbsDom([n]^m)$: $\mathcal{Q} \sqsubseteq \gamma_{S\to[n]^m}(\mathcal{P})$ iff $\alpha_{[n]^m\to S}(\mathcal{Q}) \sqsubseteq \mathcal{P}$.*

Proof. From Theorem 5.1, we have $\mathcal{Q} \sqsubseteq \gamma_{S\to T}(\mathcal{P})$ iff $\alpha_{T\to S}(\mathcal{Q}) \sqsubseteq \mathcal{P}$, provided $\mathcal{Q} = \alpha_{[n]^m\to T}(\mathcal{R})$ for some $\mathcal{R} \in AbsDom([n]^m)$. Now let $T = [n]^m$ and notice that $\alpha_{[n]^m\to[n]^m}$ is the identity function, so the theorem follows. □

A Galois connection is a cornerstone of any abstract interpretation. The reason lies in the expression $\alpha_{[n]^m\to S}(\mathcal{Q}) \sqsubseteq \mathcal{P}$. Specifically, while $\alpha_{[n]^m\to S}(\mathcal{Q})$ is a good abstraction of $\mathcal{Q}$, in practice we can compute only an approximation of $\alpha_{[n]^m\to S}(\mathcal{Q})$, which we can call $\mathcal{P}$. Given that $\mathcal{Q}$ is a density matrix, and $\alpha_{[n]^m\to S}(\mathcal{Q})$ is a tuple of approximately reduced density matrices, we can say that $\mathcal{P}$ is a tuple of approximately reduced density matrices. When we know $\alpha_{[n]^m\to S}(\mathcal{Q}) \sqsubseteq \mathcal{P}$, the Galois connection tells us that the relationship between the density matrix $\mathcal{Q}$ and the concretization of $\mathcal{P}$ is $\mathcal{Q} \subseteq \gamma_{S\to[n]^m}(\mathcal{P})$. The Galois connection will be crucial in our proof that assertion checking is correct (Theorem 8.1).

Theorem 5.3. *If $S \trianglelefteq T$ and $T \trianglelefteq R$, then $\alpha_{T\to S} \circ \alpha_{R\to T}(\mathcal{R}) \sqsubseteq \alpha_{R\to S}(\mathcal{R})$. If $R = [n]^m$, we get this special case with a stronger property: $\alpha_{T\to S} \circ \alpha_{[n]^m\to T} = \alpha_{[n]^m\to S}$.*

Theorem 5.4. *If $S \trianglelefteq T$ and $T \trianglelefteq R$, then $\gamma_{T\to R} \circ \gamma_{S\to T} = \gamma_{S\to R}$.*

5.3 Example

Consider again our running example in Section 2 and recall that it uses 3 qubits. Now we will take the first step towards abstract interpretation of that example, and we will use abstract states that focus on two qubits. Specifically, let S be a tuple that consists of every subset of $[3]$ with two elements, that is, $S = \{\{0,1\}, \{0,2\}, \{1,2\}\}$. Thus, an element $\mathcal{P} \in AbsDom(S)$ is a tuple of 4×4 projections $P_{i,j}$:

$$\mathcal{P} \quad = \quad (P_{0,1},\ P_{0,2},\ P_{1,2})$$

In Figure 1, the initial state is $v = |000\rangle$, which means that the initial density matrix is:

$$\rho = vv^\dagger = |000\rangle(|000\rangle)^\dagger = |000\rangle\langle 000|.$$

Now we can calculate the initial abstract state by applying $\alpha_{[3]\to S}$.

$$\begin{array}{ll} & \alpha_{[3]\to S}(\rho) \\ = & (\mathrm{supp}(\mathrm{Tr}_2\ \rho), \mathrm{supp}(\mathrm{Tr}_1\ \rho), \mathrm{supp}(\mathrm{Tr}_0\ \rho)) \\ = & (\mathrm{supp}(|00\rangle\langle 00|), \mathrm{supp}(|00\rangle\langle 00|), \mathrm{supp}(|00\rangle\langle 00|)) \\ = & (|00\rangle\langle 00|, |00\rangle\langle 00|, |00\rangle\langle 00|) \\ = & \left(\begin{pmatrix} 1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{pmatrix}, \begin{pmatrix} 1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{pmatrix}, \begin{pmatrix} 1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{pmatrix} \right) \end{array}$$

6 Abstract Operations

6.1 Definitions

Consider $S = (s_1, \cdots, s_m)$ where $s_i \subseteq [n]$ and $AbsDom(S)$. For a unitary matrix U that a program applies to the qubit list F, we use s_F to denote the set of qubits in F. We define an abstract operation $U_F^\sharp$:

$$U_F^\sharp \quad : \quad AbsDom(S) \ \to \ AbsDom(S)$$

The first step is to define a finer set $T = (t_1, \cdots, t_m)$, where $t_i = s_i \cup s_F$. We can check easily that $S \trianglelefteq T$. For an abstract state $\mathcal{Q} = (Q_{t_1}, \cdots, Q_{t_m}) \in AbsDom(T)$, we define

$$U_F^{cg}(\mathcal{Q}) = (UQ_{t_1}U^\dagger, \cdots, UQ_{t_m}U^\dagger),$$

where U is regarded as a unitary matrix applied on t_j in each $UQ_{t_j}U^\dagger$ since $s_F \subseteq t_j$ for all t_j. This means that before multiplications, we expand U appropriately via tensor product.

Now we can define an abstract step:

$$U_F^\sharp \quad = \quad \alpha_{T\to S} \circ U_F^{cg} \circ \gamma_{S\to T}$$

We avoid picking $T = [n]^m$ because the computation of $\gamma_{S\to[n]^m}$ would cost exponential time and space, and so would U_F^{cg} and $\alpha_{[n]^m\to S}$.

Our definition of $U_F^\sharp$ uses a kind of partial concretization that for each of the m abstractions enriches s_i to $s_i \cup s_F$, where s_F is the set of qubits affected by U. The step of partial concretization is exactly the *focus* operation that was pioneered in three-valued-logic-based shape analysis [50]. The idea is to move to a richer domain that loses no precision for the operation at hand and then drop back to the impoverished domain. So, F for Focus!

Our definition $t_i = s_i \cup s_F$ can be modified in many ways that also make our results hold; we only require $s_i \subseteq t_i$. Intuitively, our choice of $s_i \cup s_F$ is just "concrete enough" to contain the distinct qubits used by U. We see this as a design point in the trade-off between precision and efficiency. Specifically, if we pick a "more concrete" set, precision goes up and efficiency goes down. The knob that we do use to tune precision is the choice of S in $AbsDom(S)$.

6.2 Properties

In this paper, the quantum state ρ is always an n-qubit projection.

Proposition 6.1. *$\forall Q_1, Q_2 \in AbsDom(T)$: if $Q_1 \sqsubseteq Q_2$, then $U^{cg}(Q_1) \sqsubseteq U^{cg}(Q_2)$.*

Lemma 6.1. $U^{cg}(\alpha_{[n]^m \to T}(\rho)) = \alpha_{[n]^m \to T}(U\rho\, U^\dagger)$.

Proof. Let $Q = \alpha_{[n]^m \to T}(\rho)$ and

$$\begin{array}{rcl} Q & = & (Q_{t_1}, \cdots, Q_{t_m}) \\ Q_{t_j} & = & \mathrm{supp}(\mathrm{Tr}_{[n]^m \setminus t_j} \rho). \end{array}$$

We have

$$\begin{array}{rcll} U^{cg}(Q) & = & (UQ_{t_1}U^\dagger, \cdots, UQ_{t_m}U^\dagger) & \\ UQ_{t_1}U^\dagger & = & U\ \mathrm{supp}(\mathrm{Tr}_{[n]^m \setminus t_j} \rho)\ U^\dagger & \\ & = & \mathrm{supp}(\mathrm{Tr}_{[n]^m \setminus t_j}(U\rho\, U^\dagger)) & \text{(Lem. 4.2 + 4.5)} \end{array}$$

□

Lemma 6.2. *If $\alpha_{[n]^m \to S}(\rho) \sqsubseteq \mathcal{P}$, then $\alpha_{[n]^m \to S}(U\rho\, U^\dagger) \sqsubseteq U^\sharp(\mathcal{P})$.*

Proof.

$$\begin{array}{lrcl} & \alpha_{[n]^m \to S}(\rho) & \sqsubseteq & \mathcal{P} \\ \Longrightarrow & \alpha_{T \to S} \circ \alpha_{[n] \to T}(\rho) & \sqsubseteq & \mathcal{P} \\ \Longrightarrow & \alpha_{[n]^m \to T}(\rho) & \sqsubseteq & \gamma_{S \to T}(\mathcal{P}) \\ \Longrightarrow & U^{cg}(\alpha_{[n]^m \to T}(\rho)) & \sqsubseteq & U^{cg} \circ \gamma_{S \to T}(\mathcal{P}) \\ \Longrightarrow & \alpha_{[n]^m \to T}(U\rho\, U^\dagger) & \sqsubseteq & U^{cg} \circ \gamma_{S \to T}(\mathcal{P}) \\ \Longrightarrow & \alpha_{T \to S} \circ \alpha_{[n]^m \to T}(U\rho\, U^\dagger) & \sqsubseteq & \alpha_{T \to S} \circ U^{cg} \circ \gamma_{S \to T}(\mathcal{P}) \\ \Longrightarrow & \alpha_{[n]^m \to S}(U\rho\, U^\dagger) & \sqsubseteq & U^\sharp(\mathcal{P}) \end{array}$$

where the first $\Longrightarrow$ uses Theorem 5.3; the second $\Longrightarrow$ uses Theorem 5.1; the third $\Longrightarrow$ uses Proposition 6.1; the fourth $\Longrightarrow$ uses Lemma 6.1; the fifth $\Longrightarrow$ uses Lemma 5.1; the last $\Longrightarrow$ uses Theorem 5.3 and the def. of $U^\sharp$. □

Theorem 6.1 (Abstract operations are monotonic)**.** *$\forall \mathcal{P}_1, \mathcal{P}_2 \in AbsDom(S)$: if $\mathcal{P}_1 \sqsubseteq \mathcal{P}_2$, then $U^\sharp(\mathcal{P}_1) \sqsubseteq U^\sharp(\mathcal{P}_2)$.*

Proof. Combine Lemma 5.1, Lemma 5.2, and Proposition 6.1. □

From the discussion in Section 5, we know that $\alpha_{[n] \to S}(\rho) \sqsubseteq \mathcal{P}$ is a key relationship. Now we will prove that computation steps preserve this relationship.

The initial concrete state of computation is $|0\rangle^{\otimes n}$, and the initial abstract state is

$$\mathcal{P}_0 = \alpha_{[n]^m \to S}(|0\rangle^{\otimes n}).$$

Now we can apply Lemma 6.2 repeatedly to get a relationship between the final concrete state and the final abstract state.

Theorem 6.2. *If the final state of the computation is v, the final state of the abstract interpretation is $\mathcal{P} \in AbsDom(S)$, and we define $\rho = vv^\dagger$, then $\alpha_{[n]^m \to S}(\rho) \sqsubseteq \mathcal{P}$.*

Proof. Suppose the computation has z steps. Thus, the abstract interpretation also has z steps. Let us write the program as the sequence $U_1 \ldots U_z$, where each U_i is a unitary matrix that operates on the qubits in the set F_i. We will use ρ_i to denote the concrete state (as a density matrix) after i steps, so we have $\rho_{i+1} = U_{i+1}\ \rho_i\ U_{i+1}^\dagger$. Similarly, we will use $\mathcal{P}_i$ to denote the abstract state after i steps, so we have $\mathcal{P}_{i+1} = U_{F_{i+1}}^\sharp(\mathcal{P}_i)$. We will prove that for all i, where $0 \le i \le z$, that $\alpha_{[n]^m \to S}(\rho_i) \sqsubseteq \mathcal{P}_i$.

We proceed by induction on the number of execution steps. In the base case of $i = 0$, we have that by definition $\mathcal{P}_0 = \alpha_{[n]^m \to S}(\rho_0)$, so $\alpha_{[n]^m \to S}(\rho_0) \sqsubseteq \mathcal{P}_0$.

In the induction step, suppose we have $\alpha_{[n]^m \to S}(\rho_i) \sqsubseteq \mathcal{P}_i$. Now we calculate

$$\begin{array}{rcll} \alpha_{[n]^m \to S}(\rho_{i+1}) & = & \alpha_{[n]^m \to S}(U_{i+1}\ \rho_i\ U_{i+1}^\dagger) & \text{(def. } \rho_{i+1}) \\ & \sqsubseteq & U_{F_{i+1}}^\sharp(\mathcal{P}_i) & \text{(Lem. 6.2)} \\ & = & \mathcal{P}_{i+1} & \text{(def. } \mathcal{P}_{i+1}) \end{array}$$

This completes the induction proof. We have shown that $\alpha_{[n]^m \to S}(\rho_z) \sqsubseteq \mathcal{P}_z = \mathcal{P}$. □

When we combine Theorem 6.2 and Theorem 5.2, we have a full-fledged framework for abstract interpretation of quantum programs. In the next section, we show an approach to assertion checking that in conjunction with abstract interpretation leads to a powerful tool.

6.3 Example

Let us consider again the program in Figure 1 and carry out an abstract interpretation. In Section 5.3 we calculated the initial abstract state, and now we can calculate the rest of the abstract states, see Figure 2. Here, $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, as defined in Section 3.

We can check easily that each 4×4 matrix is a projection.

Notice that applying $H(0)$ affects only $P_{0,1}$ and $P_{0,2}$, but leaves $P_{1,2}$ unchanged. Intuitively, the reason why $P_{1,2}$ is unchanged is that $0 \notin \{1, 2\}$. In contrast, applying CNOT(1,2)

$$
\begin{array}{llcccccl}
\text{Initial state:} & (& |00\rangle\langle 00| & , & |00\rangle\langle 00| & , & |00\rangle\langle 00| &) \\
\text{Apply H}^\sharp(0)\text{:} & (& |0+\rangle\langle 0+| & , & |0+\rangle\langle 0+| & , & |00\rangle\langle 00| &) \\
\text{Apply H}^\sharp(1)\text{:} & (& |++\rangle\langle ++| & , & |0+\rangle\langle 0+| & , & |0+\rangle\langle 0+| &) \\
\text{Apply X}^\sharp(2)\text{:} & (& |++\rangle\langle ++| & , & |1+\rangle\langle 1+| & , & |1+\rangle\langle 1+| &) \\
\text{Apply CNOT}^\sharp(1,2)\text{:} & (& |0+\rangle\langle 0+| + |1+\rangle\langle 1+| & , & |0+\rangle\langle 0+| + |1+\rangle\langle 1+| & , & \frac{1}{2}(|01\rangle + |10\rangle)\,(|01\rangle + |10\rangle)^\dagger &) \\
\text{Apply CNOT}^\sharp(0,2)\text{:} & (& |++\rangle\langle ++| + |--\rangle\langle --| & , & |++\rangle\langle ++| + |--\rangle\langle --| & , & |++\rangle\langle ++| + |--\rangle\langle --| &) \\
\text{Apply H}^\sharp(0)\text{:} & (& |+0\rangle\langle +0| + |-1\rangle\langle -1| & , & |+0\rangle\langle +0| + |-1\rangle\langle -1| & , & |++\rangle\langle ++| + |--\rangle\langle --| &) \\
\text{Apply H}^\sharp(1)\text{:} & (& |00\rangle\langle 00| + |11\rangle\langle 11| & , & |+0\rangle\langle +0| + |-1\rangle\langle -1| & , & |+0\rangle\langle +0| + |-1\rangle\langle -1| &) \\
\text{Apply H}^\sharp(2)\text{:} & (& |00\rangle\langle 00| + |11\rangle\langle 11| & , & |00\rangle\langle 00| + |11\rangle\langle 11| & , & |00\rangle\langle 00| + |11\rangle\langle 11| &)
\end{array}
$$

Figure 2. The abstract states of GHZ for 3 qubits.

$$
\begin{array}{llcccccl}
\text{Initial state:} & (& |00\rangle\langle 00| & , & |00\rangle\langle 00| & , & |00\rangle\langle 00| &) \\
\text{Apply } \gamma_{S\to T}\text{:} & (& |00\rangle\langle 00| & , & |00\rangle\langle 00| & , & |000\rangle\langle 000| &) \\
\text{Apply } H^{cg}_{\{0\}}\text{:} & (& |0+\rangle\langle 0+| & , & |0+\rangle\langle 0+| & , & |00+\rangle\langle 00+| &) \\
\text{Apply } \alpha_{T\to S}\text{:} & (& |0+\rangle\langle 0+| & , & |0+\rangle\langle 0+| & , & |00\rangle\langle 00| &)
\end{array}
$$

Figure 3. The application of $\mathrm{H}^\sharp(0)$ to the initial state.

affects all three matrices. Intuitively, the reason is that all three matrices have either 1 or 2 or both 1,2 in the subscript sets.

Let us examine the first application of $H(0)$ in detail. We have $S = (\{0,1\},\{0,2\},\{1,2\})$ and $F = \{0\}$, so we get $T = (\{0,1\}\cup\{0\},\{0,2\}\cup\{0\},\{1,2\}\cup\{0\}) = (\{0,1\},\{0,2\},\{0,1,2\})$. Figure 3 shows the initial state, the tuple of matrices after applying $\gamma_{S\to T}$, then after applying $H^{cg}_{\{0\}}$, and finally after applying $\alpha_{T\to S}$. Notice that the final tuple of matrices in Figure 3 matches the tuple of matrices in Figure 2 after we apply $\mathrm{H}^\sharp(0)$. Notice also that two of the matrices are of size 8×8, which is because they are at the position in the tuple for which the corresponding entry in T is $\{0,1,2\}$.

Now let us zoom in closer and look at the first matrix in each row of Figure 3. How do we calculate the first matrix in the second row? The set for that matrix is $\{0,1\}$.

We begin with $\gamma_{S\to T}$. The definition of $\gamma_{S\to T}$ calls for finding elements of T that each is a subset of $\{0,1\}$. We find only one such set, namely $\{0,1\}$ itself. So, going through the definition of $\gamma_{S\to T}$, we see that $Q_{0,1} = P_{0,1}$.

Next we have U^{cg}_0, and here U is H. We need to expand H to work with a 4×4 matrix, and in this case, the expansion is $H\otimes I$. Then we can multiply $H\otimes I$ with the first matrix in the second row of Figure 3, which gives us the first matrix in the third row.

Finally, we have $\alpha_{T\to S}$. The definition of $\alpha_{T\to S}$ calls for finding elements of T such that each is a superset of $\{0,1\}$. We find two such sets, namely $\{0,1\}$ itself and $\{0,1,2\}$. So, we take the first and the third matrix in the third row and get ready for the detailed computation of $\alpha_{T\to S}$. This computation intersects two matrices that each is obtained by computing first a trace out and then the support of the matrix. For the case of the first matrix in the third row, we see that the two sets are equal so no trace out will happen, after which computing the support has no effect. For the case of the third matrix in the third row, we first trace out 2 and then compute the support. Finally, we compute the intersection of those matrices, which gives the first matrix in the first row.

One may use a simpler way to deal with a single unitary matrix: apply the single qubit unitary matrix on corresponding two-qubit projections. The reason that we do not use this is as follows. At each step, the abstract state may contains some redundant information, e.g., $(|00\rangle\langle 00| + |11\rangle\langle 11|, |00\rangle\langle 00| + |11\rangle\langle 11|, |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 10| + |11\rangle\langle 11|)$. The $\cap$ has the potential to eliminate some of the redundant information, that is, obtaining a more compact abstract state. For the mentioned example, a more compact form is $(|00\rangle\langle 00| + |11\rangle\langle 11|, |00\rangle\langle 00| + |11\rangle\langle 11|, |00\rangle\langle 00| + |11\rangle\langle 11|)$.

Notice that the example has $n = 3$ qubits and considers all the possible local projections that focus on $k = 2$ qubits. Thus, we work with n-choose-k = 3-choose-2 = 3 local projections. In Section 9, we will do abstract interpretation on GHZ for $n = 300$ and $k = 2$, leading to 300-choose-2 = 44,850 local projections.

6.4 Space and Time Requirements

Space Requirements. Let us compare the space needed for a simulation to store a concrete state and the space needed for abstract interpretation to store an abstract state.

For a program with n qubits, the concrete state is a vector with 2^n complex numbers.

For an estimate of the size of an abstract state, suppose we have picked a k and then picked S to contain sets of only size k. In this case, each abstract state is of size $O(|S| \times (2^k \times 2^k))$. However, we need more space than that. The reason has to do with the way the abstract steps work. In our benchmarks, the gates are 1-qubit gates, 2-qubit gates, and 3-qubit gates so our worst case is 3-qubit gates. Thus, during the execution of an abstract step, we use matrices of size at most $(2^{k+3} \times 2^{k+3})$, so the total need for space is $O(|S| \times (2^{k+3} \times 2^{k+3}))$.

For example, as we will discuss in Section 9, for Grover's algorithm with 300 qubits, we use $k = 5$ and $|S| = 148$. Each concrete state is of size 2^{300} complex numbers which are more than the number of atoms in the known universe. In contrast, each abstract state is of a worst-case size of $|S| \times (2^{k+3} \times 2^{k+3}) < 10$ million complex numbers. Thus, we can easily store an abstract state on a laptop.

Time Requirements. In Section 9, we will show measurements that suggest that at scale, we can focus the analysis of execution time on the transformation step of applying U_F^{cg}. The reason is that the step of applying $\alpha_{T\to S}$ is faster, while the step of applying $\gamma_{S\to T}$ is within a factor of 2 slower.

The main work of applying U_F^{cg} is to multiply three $(2^{k+3} \times 2^{k+3})$ matrices, which will take $2 \times (2^{k+3})^3 = 2^{3k+10}$ multiplications of complex numbers. So, in a program p with $|p|$ gates and use of 1-qubit gates, 2-qubit gates, and 3-qubit gates, the back-of-the-envelope worst-case running time is

$$O(|p| \times 8^k).$$

7 Assertion Checking

We will consider a particular form of assertions that we have found to be useful for a variety of quantum programs. Those assertions are of a form that can be easily mapped to a projection, which makes it easy to work within our setting.

7.1 Definitions

Consider a program with n qubits. An assertion A is a span of two vectors of a particular form:

$$A \quad = \quad \mathrm{span}\{|a_1\rangle|a_2\rangle \cdots |a_n\rangle, |b_1\rangle|b_2\rangle \cdots |b_n\rangle\}$$

For an assertion A, we can define a projection $\mathrm{proj}(A)$ onto space defined by A. Specifically, $\mathrm{proj}(A)$ is a rank 1 or rank 2 projection of n-qubit system such that

$$\mathrm{proj}(A)|a_1\rangle|a_2\rangle \cdots |a_n\rangle = |a_1\rangle|a_2\rangle \cdots |a_n\rangle,$$
$$\mathrm{proj}(A)|b_1\rangle|b_2\rangle \cdots |b_n\rangle = |b_1\rangle|b_2\rangle \cdots |b_n\rangle.$$

Using an appropriately chosen S and $\mathcal{P} \in AbsDom(S)$, we check $\mathcal{P} \sqsubseteq \alpha_{[n]\to S}(\mathrm{proj}(A))$. As we will show in the following section, if the final abstract state satisfies this property, then the final concrete state satisfies the assertion A.

7.2 Properties

Our form of assertions has a delightful property that we express as Lemma 7.1 below. Intuitively, if we map the span of two vectors to the corresponding giant projection, then abstract the giant projection to an abstract state, and finally, concretize the abstract state back to a giant projection, we get exactly the original giant projection! In the following section, we will combine this property with our framework for abstract interpretation.

Before that, we introduce a concept about the "connectivity" of S. $S = (s_1, \cdots, s_m)$ with $s_i \subseteq [n]$ is called "connected" if for any $k, l \in [n]$, there exists integer t and a sequence $a_0, \cdots, a_t \in [n]^t$ such that $a_0 = k$, $a_t = l$ and for any $0 \le i < t$, both $a_i, a_{i+1} \in s_{r_i}$ for some $1 \le r_i \le m$.

Lemma 7.1. *For an assertion*

$$A \quad = \quad \mathrm{span}\{|a_0 a_1 \cdots a_{n-1}\rangle, |b_0 b_1 \cdots b_{n-1}\rangle\},$$

we have $\mathrm{proj}(A) = \gamma_{S\to[n]}(\alpha_{[n]\to S}(\mathrm{proj}(A)))$ *holds if* S *is "connected".*

7.3 Example

For the GHZ benchmark for 3 qubits, we saw in Figure 1 that our assertion is that the final state is in

$$A_{GHZ3} \quad = \quad \mathrm{span}\{ \quad |000\rangle, \quad |111\rangle \quad \}$$

In Section 5.3, we picked $S = \{\{0,1\}, \{0,2\}, \{1,2\}\}$. Now we can find $\alpha_{[3]\to S}(\mathrm{proj}(A_{GHZ3}))$ equals

$$\left(\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \right)$$

We see that $\alpha_{[3]\to S}(\mathrm{proj}(A_{GHZ3}))$ is equal to the final abstract state $\mathcal{P}$ in Figure 2, that is, we have $\alpha_{[3]\to S}(\mathrm{proj}(A_{GHZ3})) = \mathcal{P}$, so $\mathcal{P} \sqsubseteq \alpha_{[3]\to S}(\mathrm{proj}(A_{GHZ3}))$.

8 Putting it All Together

Now we are ready to combine the Galois connection (Theorem 5.2), the relationship between the final concrete state and the final abstract state (Theorem 6.2), and the property of our form of assertions (Lemma 7.1). The result is that assertion checking is correct: if the final abstract state satisfies the assertion, then the final concrete state satisfies the assertion, too.

Theorem 8.1 (Assertion Checking is Correct)**.** *For assertion A defined in Section 7, if the final state of the computation is v, the final state of the abstract interpretation is $\mathcal{P} \in AbsDom(S)$, and $\mathcal{P} \sqsubseteq \alpha_{[n]\to S}(\mathrm{proj}(A))$, then $v \in A$.*

Proof. Define $\rho = vv^\dagger$. We reason as follows:

$$\begin{array}{llll} & \alpha_{[n]\to S}(\rho) & \sqsubseteq & \mathcal{P} & (\text{Thm } 6.2) \\ \text{iff} & \rho & \subseteq & \gamma_{S\to[n]}(\mathcal{P}) & (\text{Thm } 5.2) \\ & & \subseteq & \gamma_{S\to[n]}(\alpha_{[n]\to S}(\mathrm{proj}(A))) & \\ & & = & \mathrm{proj}(A) & (\text{Lem } 7.1) \end{array}$$

where in the third line, we use $\mathcal{P} \sqsubseteq \alpha_{[n]\to S}(\mathrm{proj}(A))$ and Lemma 5.2.

Next, notice that $\rho(v) = (vv^\dagger)v = v(v^\dagger v) = v1 = v$. From $\rho \subseteq \mathrm{proj}(A)$ and $\rho(v) = v$, we have $(\mathrm{proj}(A))(v) = v$, hence $v \in A$. □

Notice that in the proof of Theorem 8.1, we use Lemma 7.1. The need for Lemma 7.1 is a limitation: it restricts the form of assertions. However, our assertions are useful because

Program	n	#gates	k	$\|S\|$	time (s)	assertion
BV	50	150	2	1,225	38	✓
	100	300	2	4,950	317	✓
	150	450	2	11,175	1,095	✓
	200	600	2	19,900	2,626	✓
	250	750	2	31,125	5,244	✓
	300	900	2	44,850	9,059	✓
GHZ	50	149	2	1,225	39	✓
	100	299	2	4,950	318	✓
	150	449	2	11,175	1,123	✓
	200	599	2	19,900	2,854	✓
	250	749	2	31,125	5,497	✓
	300	899	2	44,850	8,959	✓
Grover	63	222	5	30	6,129	✓
	127	446	5	62	30,693	✓
	255	894	5	126	114,117	✓
	300	1,052	5	148	166,633	✓

Figure 4. Measurements.

the idea of working in a two-dimensional subspace has been used extensively in quantum algorithm design. We leave to future work to generalize Lemma 7.1.

9 Experimental Results

Questions and claims. Our experimental evaluation answers three questions about our approach to quantum abstract interpretation. The questions and our claims are as follows.

1. Is our approach scalable? *Yes, it scales to programs with 300 qubits.*
2. Is our approach useful? *Yes, it checks the assertions in three families of benchmark programs.*
3. Is our approach flexible? *Yes, it enables users to change the abstract domain easily.*

In the remainder of this section, we show how our evaluation supports our claims.

Implementation. We have implemented our approach in 2,350 lines of Java, on top of an existing matrix library and an automatically generated parser. Our implementations of tensor products, trace out, and intersection are straightforward, and we compute the support of a matrix using the Gram-Schmidt process. Our implementation includes validity checks of the produced matrices, which optionally can be done at every step of an abstract interpretation. Specifically, we check that every matrix is a nonzero projection.

Platform. We ran all experiments on a MacBook Pro with an Intel Core i7 processor with 2.2 GHz processor clock frequency and 16 GB main memory.

Benchmarks. We have a benchmark suite of 16 quantum programs that can be divided into three families, see Figure 4. The column labeled n shows the number of qubits. Each program uses between 50 and 300 qubits and has between 149 and 1,052 gates.

The first family of benchmarks implements the Bernstein-Vazirani algorithm [10], which is labeled *BV* in Figure 4. for 50, 100, 150, 200, 250, and 300 qubits. For n qubits, the Bernstein-Vazirani algorithm solves the following problem. The input is a representation of a black-box linear function $f : \{0,1\}^n \to \{0,1\}$, where $f(x) = a \times x + b$. Here, a is an unknown bit string of length n, $\times$ is inner product mod 2, $+$ is addition mod 2, and b is an unknown single bit. The goal is to output a. The amazing property of the Bernstein-Vazirani algorithm is that it determines a after a *single* invocation of the representation of f. In contrast, a classical computer needs to do n invocations of f to determine a. Our assertion about the final state before the measurement is that it is in

$$\text{span}\ \{\ |a\rangle\ \}$$

Given that every state is a unit vector, we see that if the assertion is correct, then the measurement will produce a. Thus, for Bernstein-Vazirani our assertion is sufficient to guarantee that the algorithm is correct.

The second family of benchmarks implements the GHZ algorithm for 50, 100, 150, 200, 250, and 300 qubits. Figure 1 shows how GHZ circuit works for 3 qubits. For n qubits, consisting of $2n - 1$ H gates, $n - 1$ CNOT gates, and one X gate, GHZ produces a final state before measurement of the form $\frac{1}{\sqrt{2}}(|00..0\rangle - |11..1\rangle)$. Thus, after measurement, we get $|00..0\rangle$ with probability 0.5 and we get $|11..1\rangle$ with probability 0.5. The amazing property of the GHZ algorithm is that it entangles n qubits, either as "all zeros" or as "all ones". This enables a wide range of quantum applications, including quantum teleportation, which in turn enables the quantum internet. Our assertion is that the final state before the measurement is in

$$\text{span}\ \{\ |00..0\rangle,\ |11..1\rangle\ \}$$

Figure 1 shows what the assertion looks like for 3 qubits. Thus, for GHZ our assertion is sufficient to guarantee that we will get one of the two expected vectors. However, our assertion is silent about the fact that the two vectors are equally likely so the assertion implies partial correctness.

The third family of benchmarks implements Grover's algorithm [33] for 63, 127, 255, and 300 qubits. The input to Grover's algorithm is a black-box predicate f that has a domain of size 2^n and that returns true on a single input; the goal is to find that input. The amazing property of Grover's algorithm is that it finds that single input after approximately $\sqrt{2^n}$ invocations of f, with high probability. In contrast, a classical computer needs to do $2^n - 1$ invocations of f, in the worst case. Grover's algorithm works as follows. The algorithm uses a collection D of n qubits that each, initially, is $|0\rangle$. Then the algorithm proceeds as follows.

1. Apply H to each qubit in D.
2. Repeat { apply G to D } approximately $\sqrt{2^n}$ times.
3. Measure D and output the result.

We skip the details of G and focus on the repeat-loop. Intuitively, each iteration increases the probability that the measurement will produce the desired value, with the peak after approximately $\sqrt{2^n}$ iterations. The loop has a geometric interpretation that says that before and after each iteration, D is always in a particular two-dimensional subspace, spanned by two vectors $|A\rangle$, $|B\rangle$ that can be defined easily [33]. We can express this property as the loop invariant that D is in span { $|A\rangle$, $|B\rangle$ } For example, if the single input for which f returns 1 is the bit vector $00..0$, then the loop invariant is

$$\text{span}\ \{\ |00..0\rangle,\ |{+}{+}..{+}\rangle\ \}$$

Given such a loop invariant, we see that the final state before the measurement is also in this span. We can show that the above is a loop invariant by checking three properties: 1) initialization establishes the invariant, 2) if $D = |00..0\rangle$, then after execution of the loop body, D is in the above span, and 3) if $D = |{+}{+}..{+}\rangle$, then after execution of the loop body, D is in the above span. The reason why those checks are sufficient to establish that the above is a loop invariant is that the loop body is a linear function G. So, if both $G(|00..0\rangle)$ and $G(|{+}{+}..{+}\rangle)$ are in the above span, then G applied to any vector in the above span is in the above span. In summary, our assertion for Grover's algorithm is the algorithm's loop invariant, which implies partial correctness. The most time-consuming check is (3), and Figure 4 reports the time to check (3).

In Grover's algorithm, the implementation of G comes with some degrees of freedom. While we skip the details here, we note that for 63, 127, and 255 qubits we used a "tree" implementation of G, while for 300 qubits we used a "linear" implementation of G. We did this to show that our approach can handle different styles of quantum programming.

The Grover benchmarks are the most challenging for abstract interpretation. The reason is that both the Bernstein-Vazirani benchmarks and the GHZ benchmarks use only so-called Clifford gates. Such benchmarks can be simulated in polynomial time on a probabilistic classical computer [31]. In contrast, our Grover examples also use Toffoli gates, which are outside the class of Clifford gates.

Measurements. Figure 4 shows our measurements for running our tool on our 16 quantum programs. Our tool successfully checked the assertion in every program.

For Bernstein-Vazirani and for GHZ, we used $k = 2$, that is, local projections onto 2 qubits, which means that every entry in S has 2 elements. Specifically, for each program, the set S, from which we define $AbsDom(S)$, consisted of every pair of 2 qubits. For example, for 50 qubits, we define S to contain all 50-choose-2 = 1,225 pairs of 2 qubits.

For Grover, we use $k = 5$, that is, local projections onto 5 qubits, which means that every entry in S has 5 elements. However, we restricted S to contain

sets that each overlaps with the qubits used by at least two 3-qubit gates in the program.

This turns out to be a small number. For example, for our version of Grover that works on 300 qubits, S contains just 148 sets of each 5 qubits.

Our implementation is unoptimized and we ran it on a laptop (a MacBook Pro); the execution times range between 38 seconds and 166,633 seconds (2 days).

Is our approach scalable? Yes, it scales to programs with 300 qubits. This means that our approach scales to programs with a state space that has more complex numbers than the number of atoms in the known universe. Our approach goes way beyond the 50 qubits that is the limit for quantum simulation on current supercomputers.

Is our approach useful? Yes, it checks the assertions in three families of benchmark programs. Indeed, our approach successfully checked the assertions in BV, GHZ, and Grover. Our results are highly encouraging: even for programs for a larger quantum computer that we have today, we can gain confidence that they satisfy key correctness criteria, *ahead* of running on the quantum computer itself.

We have done experiments with additional programs and we have yet to find a case where our implementation was unable to verify the specified assertion.

Is our approach flexible? Yes, it enables users to change the abstract domain easily. Indeed, our implementation has several command-line parameters that enable easy change of the abstract domain. In principle, our implementation supports any of the abstract domains defined in this paper, but in practice, scalability limits which ones we can try. The largest abstract domain that we have tried is based on 44,850 sets of qubits. Additionally, the largest local projections that we have tried use five qubits. This means that our approach internally computes with matrices of complex numbers that are up to size $2^{5+3} \times 2^{5+3} = 256 \times 256$.

For Grover, we can think of the highlighted phrase above (under Measurements) as the specification of dependency analysis. The idea is to focus our abstract domain on how gates depend on each other. For example, suppose a gate U_1 produces a result in qubit variable i, which then is used by a gate U_2. In this case, we will do well to consider the qubits used by U_1 and U_2 together. So, we take the union of the qubits used by U_1 and the qubits used by U_2 and add that set to S. The result is that the abstract interpretation does a good job of tracking the flow of data. For example, in the Grover benchmark with n=63, we have the lines:

NCNCNOT(4,5,34)
[...]
CCNOT(34,35,49)

Here, NCNCNOT(4,5,34) uses the qubits $\{4, 5, 34\}$, while CCNOT(34,35,49) uses the qubits $\{34, 35, 49\}$. In more detail, for the NCNCNOT gate, qubit 34 is the target qubit, while for the CCNOT gate, qubit 34 is one of the control qubits. We take the union of those two sets and get $\{4, 5, 34, 35, 49\}$, which we then add to S. This set $\{4, 5, 34, 35, 49\}$ helps us model the data flow from the line NCNCNOT(4,5,34) to the line CCNOT(34,35,49). In the terminology of classical compilers, we can say that our choice of S makes the abstract interpretation *flow-sensitive*.

For Grover, we have tried various cases of f, k and S. First, we have found that varying f leads to the same verification results. Second, for some cases of Grover that work on less than 63 qubits, we have found that $k = 4$ and sometimes even $k = 3$ are sufficient to check the assertions. However, for the benchmarks that we use in this paper, $k = 5$ appears to be necessary. In particular, for Grover and $n \geq 63$ and $k < 5$, all our attempts at assertion checks actually failed, even though the assertions are true. On the positive side, we conjecture that for cases of Grover with more than 300 qubits, $k = 5$ will continue to be sufficient.

Where was the time spent? The implementation does initialization, abstract steps, and validity checks. The initialization and validity checks take a total of fewer than 4 seconds in all cases so let us focus on the abstract steps. As explained in Section 6, each abstract step is of the form: $\alpha_{T\to S} \circ U_F^{cg} \circ \gamma_{S\to T}$. Let us use the terminology that the application of $\alpha_{T\to S}$ is the *alpha* step, while the application of U_F^{cg} is the *transformation* step and the application of $\gamma_{S\to T}$ is the *gamma* step. Figure 5 shows how the time was spent on those three operations for the benchmarks with 300 qubits.

The results for Bernstein-Vazirani and for GHZ are similar. They show that the gamma step dominates, yet that all three steps take a nontrivial percentage of the time. This is because S is large so all three steps have a lot of work to do.

In contrast, the results for Grover are quite different and we see that the transformation step dominates, while the alpha step takes almost no time. The reason why the transformation step dominates that for Grover we use $k = 5$ so all the matrices are of size 256×256. This makes the matrix multiplications in the transformation step take a long time. However, for Grover, we use a small S, which has the effect that α finds little work to do. Specifically, for any element s_i in S, the execution of the alpha step will find a few supersets of s, hence little computation to do.

The above analysis shows that if we can work with a small S, then the gamma and transformation steps do almost all of the work. We leave to future work to automatically pick a small S for a given program. Classically, abstract interpretation uses widening to accelerate an analysis; perhaps widening can help adaptively decrease the size of S. Additionally, abstraction refinement may be a way to proceed after an assertion check fails. A starting point for abstraction refinement could be to let S be the set of one-qubit projections. For such an S, assertion checking will likely fail, after which we can refine S, and so on.

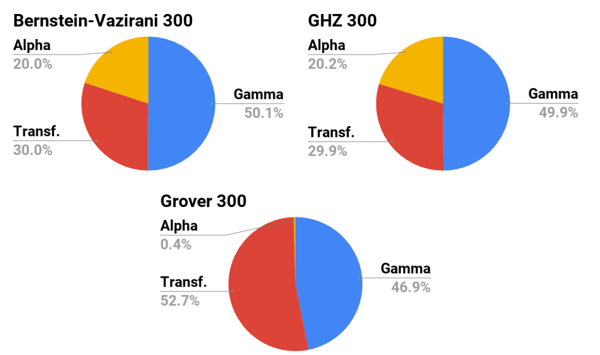


Figure 5. Where time was spent in the abstract steps.

10 Conclusion

We have shown how to do efficient abstract interpretation of quantum programs and shown that it is useful for checking assertions. Our tool has successfully processed quantum programs that use many more qubits than what can be handled by simulation. This demonstrates the effectiveness of using local projections as the core abstraction.

Our results open the door to many directions for future work that include: generalization to programs with measurement, conditionals, and loops, that is, a mix of classical and quantum computation, which leads to an efficient quantum Hoare logic [63]. This may enable abstract interpretation of hybrid algorithms such as Shor's algorithm. Other directions are choice of abstract domain, other types of assertions, multiple assertions within a single program, parallel implementation (perhaps with GPUs), and generalization from projections to local Hamiltonians.

Acknowledgments

We thank Hanru Jiang and Henry Ma for technical discussions. We also thank Henry Ma, Pratik Sathe, and the anonymous PLDI reviewers for helpful comments on a draft of the paper, and we thank our PLDI shepherd Thomas Reps for guidance. N. Yu is supported by ARC Discovery Early Career Researcher Award DE180100156 and ARC Discovery Project DP210102449. J. Palsberg is supported by the NSF QLCI program through grant number OMA-2016245.

References

[1] Ali Javadi Abhari, Arvin Faruque, Mohammad Javad Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, Fred Chong, Margaret Martonosi, Martin Suchara, Ken Brown, Massoud Pedram, and Todd Brun. 2012. *Scaffold: Quantum programming language*. Technical Report TR-934-12. Dept. of Computer Science, Princeton University NJ. ftp://ftp.cs.princeton.edu/reports/2012/934.pdf

[2] Dmitri Akatov. 2005. *The Logic of Quantum Program Verification*. Master's thesis. Oxford University Computing Laboratory. http://www.academia.edu/download/7563948/thesis-1.1.ps

[3] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernádez, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. Qiskit: An Open-source Framework for Quantum Computing. (2019). https://doi.org/10.5281/zenodo.2562110

[4] Thorsten Altenkirch and Jonathan Grattage. 2005. A functional quantum programming language. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*. IEEE, 249–258.

[5] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B. Buckley, David A. Buell, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Sean Demura, Andrew Dunsworth, Edward Farhi, Austin Fowler, Brooks Foxen, Craig Gidney, Marissa Giustina, Rob Graff, Steve Habegger, Matthew P. Harrigan, Alan Ho, Sabrina Hong, Trent Huang, L. B. Ioffe, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Cody Jones, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Seon Kim, Paul V. Klimov, Alexander N. Korotkov, Fedor Kostritsa, David Landhuis, Pavel Laptev, Mike Lindmark, Martin Leib, Erik Lucero, Orion Martin, John M. Martinis, Jarrod R. McClean, Matt McEwen, Anthony Megrant, Xiao Mi, Masoud Mohseni, Wojciech Mruczkiewicz, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Florian Neukart, Hartmut Neven, Murphy Yuezhen Niu, Thomas E. O'Brien, Bryan O'Gorman, Eric Ostby, Andre Petukhov, Harald Putterman, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Andrea Skolik, Vadim Smelyanskiy, Doug Strain, Michael Streif, Kevin J. Sung, Marco Szalay, Amit Vainsencher, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, and Leo Zhou. 2020. Quantum Approximate Optimization of Non-Planar Graph Problems on a Planar Superconducting Processor. (2020). https://arxiv.org/abs/2004.04197.

[6] Alexandru Baltag and Sonja Smets. 2004. The logic of quantum programs. In *Proceedings of the 2nd International Workshop on Quantum Programming Languages (QPL 2004)*, Peter Selinger (Ed.). 39–56.

[7] Alexandru Baltag and Sonja Smets. 2006. LQP: the dynamic logic of quantum information. *Mathematical Structures in Computer Science* 16, 3 (2006), 491–525.

[8] Gilles Barthe, Justin Hsu, Mingsheng Ying, Nengkun Yu, and Li Zhou. 2019. Coupling Techniques for Reasoning about Quantum Programs. (2019). arXiv:1901.05184

[9] P Benioff. 1980. Computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers represented by Turing machines. *J. Stat. Phys.; (United States)* 22:5, 525-532 (1 1980).

[10] Ethan Bernstein and Umesh Vazirani. 1997. Quantum Complexity Theory. *SIAM J. Comput.* 26, 5 (Oct. 1997), 1411–1473.

[11] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. 2014. Exponential Improvement in Precision for Simulating Sparse Hamiltonians. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing (STOC '14)*. Association for Computing Machinery, New York, NY, USA, 283–292.

[12] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. 2015. Hamiltonian Simulation with Nearly Optimal Dependence on all Parameters. *2015 IEEE 56th Annual Symposium on Foundations of Computer Science* (Oct 2015). https://doi.org/10.1109/focs.2015.54

[13] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549, 7671 (01 Sep 2017), 195–202. https://doi.org/10.1038/nature23474

[14] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. 2020. Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. ACM, New York, NY, USA.

[15] Garrett Birkhoff and John Von Neumann. 1936. The logic of quantum mechanics. *Annals of Mathematics* 37, 4 (1936), 823–843.

[16] G. Brassard and P. Hoyer. 97. An exact quantum polynomial-time algorithm for Simon's problem. *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems* (97), 12–23.

[17] G. Brassard, P. Hoyer, Michele Mosca, and Alain Tapp. 2002. Quantum Amplitude Amplification and Estimation. *Quantum Computation and Quantum Information* 305 (2002), 53–74.

[18] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, Martin Leib, Andrea Skolik, Michael Streif, David Von Dollen, Jarrod R. McClean, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, and Masoud Mohseni. 2020. TensorFlow Quantum: A Software Framework for Quantum Machine Learning. (2020). arXiv:quant-ph/2003.02989

[19] Olivier Brunet and Philippe Jorrand. 2004. Dynamic quantum logic for quantum programs. *International Journal of Quantum Information* 2, 01 (2004), 45–54.

[20] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. 2019. Quantum Chemistry in the Age of Quantum Computing. *Chemical Reviews* 119, 19 (2019), 10856–10915. PMID: 31469277.

[21] Rohit Chadha, Paulo Mateus, and Amílcar Sernadas. 2006. Reasoning about imperative quantum programs. *Electronic Notes in Theoretical Computer Science* 158 (2006), 19–39.

[22] Christophe Chareton, Sébastien Bardin, Fran$C$cois Bobot, Valentin Perrelle, and Benoît Valiron. 2021. An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In *European Symposium on Programming 2021: Programming Languages and Systems*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham,

148–177.

[23] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Fourth ACM Symposium on Principles of Programming Languages*. 238–252.

[24] D. Deutsch. 1985. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London Series A* 400, 1818 (July 1985), 97–117.

[25] D. Deutsch. 1989. Quantum Computational Networks. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences* 425, 1868 (1989), 73–90. http://www.jstor.org/stable/2398494

[26] Ellie D'hondt and Prakash Panangaden. 2006. Quantum weakest preconditions. *Mathematical Structures in Computer Science* 16, 3 (2006), 429–451.

[27] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. (2014). https://arxiv.org/abs/1411.4028.

[28] Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. 2007. Proof rules for the correctness of quantum programs. *Theoretical Computer Science* 386, 1-2 (2007), 151–166.

[29] Richard P. Feynman. 1982. Simulating Physics with Computers. *International Journal of Theoretical Physics* 21 (1982), 467–488.

[30] Richard P. Feynman. 1986. Quantum mechanical computers. *Foundations of Physics* 16 (1986), 507–531.

[31] Daniel Gottesman. 2008. The Heisenberg Representation of Quantum Computers. (2008). Los Alamos National Laboratory. Expanded version of a plenary speech at the 1998 International Conference on Group Theoretic Methods in Physics.

[32] Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*. ACM, New York, NY, USA, 333–342.

[33] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing (STOC '96)*. 212–219.

[34] Emily Grumbling and Mark Horowitz. 2019. *Quantum Computing: Progress and Prospects*. Technical Report doi: https://doi.org/10.17226/25196. National Academies of Sciences, Engineering, and Medicine, The National Academies Press, Washington, D.C.

[35] Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. 2020. Proving Quantum Programs Correct. (2020). arXiv:2010.01240.

[36] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021. A Verified Optimizer for Quantum Circuits. *Proc. ACM Program. Lang.* POPL'2021.

[37] Shih-Han Hung, Kesha Hietala, Shaopeng Zhu, Mingsheng Ying, Michael Hicks, and Xiaodi Wu. 2019. Quantitative Robustness Analysis of Quantum Programs. *Proc. ACM Program. Lang.* 3, POPL (2019), 31:1–31:29.

[38] Yoshihiko Kakutani. 2009. A logic for formal verification of quantum programs. In *Proceedings of the 13th Asian conference on Advances in Computer Science: information Security and Privacy (ASIAN 2009)*, Anupam Datta (Ed.). Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 79–93.

[39] Gudrun Kalmbach. 1983. *Orthomodular lattices*. Vol. 18. Academic Press.

[40] Aleks Kissinger and John van deWetering. 2020. PyZX: Large Scale Automated Diagrammatic Reasoning. *Electronic Proceedings in Theoretical Computer Science* 318, 4 (2020), 230–242.

[41] Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. 2020. Projection-Based Runtime Assertions for Testing and Debugging Quantum Programs. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 150 (Nov. 2020), 29 pages.

[42] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. 2016. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics* 18, 2 (feb 2016), 023023.

[43] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press, New York, NY, USA.

[44] Bernhard Ömer. 2003. *Structured quantum programming*. Ph.D. Dissertation. Institute for Theoretical Physics, Vienna University of Technology.

[45] Jennifer Paykin, Robert Rand, and Steve Zdancewic. 2017. QWIRE: a core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. ACM, New York, NY, USA, 846–858.

[46] Simon Perdrix. 2008. Quantum Entanglement Analysis Based on Abstract Interpretation. In *Proceedings of SAS'08, Static Analysis Symposium*. Springer-Verlag (*LNCS* 5079), 270–282.

[47] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications* 5, 1 (23 Jul 2014), 4213.

[48] Robert Rand. 2016. Verification logics for quantum programs. (2016). http://www.cs.umd.edu/~rrand/wpe.pdf

[49] Amr Sabry. 2003. Modeling Quantum Computing in Haskell. In *Proceedings of the 2003 ACM SIGPLAN Workshop on Haskell*.

[50] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. 1999. Parametric Shape Analysis via 3-Valued Logic. *ACM TOPLAS* 24, 3 (1999), 217–298.

[51] Jeff W Sanders and Paolo Zuliani. 2000. Quantum programming. In *International Conference on Mathematics of Program Construction (MPC 2000)*, Roland Backhouse and José Nuno Oliveira (Eds.). Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 80–99.

[52] Peter Selinger. 2004. Towards a quantum programming language. *Mathematical Structures in Computer Science* 14, 4 (2004), 527–586.

[53] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018 (RWDSL 2018)*. ACM, New York, NY, USA, 7:1–7:10.

[54] The Cirq Developers. 2018. quantumlib/Cirq: A python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits. (2018). https://github.com/quantumlib/Cirq.

[55] Dominique Unruh. 2019. Quantum relational Hoare logic. In *Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2019)*. ACM, New York, NY, USA.

[56] D. Wecker and K. M. Svore. 2014. LIQUi|⟩: A software design architecture and domain-specific language for quantum computing. (2014).

[57] Wikipedia. 2020. List of quantum processors. (2020). https://en.wikipedia.org/wiki/List_of_quantum_processors.

[58] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. 2019. Full-state quantum circuit simulation by using data compression. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov 2019). https://doi.org/10.1145/3295500.3356155

[59] Mingsheng Ying. 2011. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 33, 6, Article 19 (2011), 49 pages.

[60] Mingsheng Ying. 2016. *Foundations of Quantum Programming*. Morgan Kaufmann.

[61] Mingsheng Ying, Runyao Duan, Yuan Feng, and Zhengfeng Ji. 2010. Predicate transformer semantics of quantum programs. *Semantic Techniques in Quantum Computation* 8 (2010), 311–360.

[62] Nengkun Yu. 2019. Quantum Temporal Logic. *arXiv preprint arXiv:1908.00158* (2019).

[63] Nengkun Yu and Jens Palsberg. 2021. An efficient quantum Hoare logic. (2021). In preparation.

[64] Li Zhou, Nengkun Yu, and Mingsheng Ying. 2019. An Applied Quantum Hoare Logic. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*. ACM, New York, NY, USA, 1149–1162.

A Appendix: Proofs of Lemma 4.6, and Theorems and Lemmas in Sections 5+7

For any square matrix A, we write $\langle\psi|A|\psi\rangle$ to mean the inner product between $|\psi\rangle$ and $A|\psi\rangle$. A Hermitian operator A is *positive semidefinite* (resp., *positive definite*) if for all vectors $|\psi\rangle \in \mathcal{H}$, $\langle\psi|A|\psi\rangle \geq 0$. This gives rise to the *Löwner order* $\leq$ among operators:

$$A \leq B \text{ if } B - A \text{ is positive semidefinite,}$$

The *Löwner order* $\leq$ among projections is equivalent to the subset relation among linear subspaces.

Proofs of the lemmas in Section 4 can be found in textbooks on linear algebra. Here we give proofs of some of the lemmas.

Lemma A.1. *For positive semi-definite matrix A and a projection P, if* $\mathrm{supp}(A) = Q$, *then there exists $r_1, r_2 > 0$ such that*

$$r_1 Q \leq A \leq r_2 Q.$$

If $\mathrm{supp}(A) \subseteq P$, *we have some $r > 0$ such that*

$$\rho \leq rP.$$

Proof. Let the spectrum decomposition [43] of A is as

$$A = \sum_{i=0}^{m-1} \lambda_i |\psi_i\rangle\langle\psi_i|$$

where $\lambda_i > 0$ are the eigenvalues, and $|\psi_i\rangle$ is the corresponding eigenvector of λ_i. Therefore,

$$\mathrm{supp}(A) = \mathrm{span}\{|\psi_0\rangle, \cdots, |\psi_{m-1}\rangle\}.$$

Write it in the projection form, it is

$$Q := \sum_{i=0}^{m-1} |\psi_i\rangle\langle\psi_i|$$

By choosing $r_1 = \min\{\lambda_0, \cdots, \lambda_{m-1}\}$, and $r_2 = \max\{\lambda_0, \cdots, \lambda_{m-1}\}$, we know that

$$r_1 Q \leq A \leq r_2 Q.$$

If $\mathrm{supp}(A) \subseteq P$, then $Q \leq P$

$$\rho \leq r_2 Q \leq r_2 P.$$

□

Lemma A.2. *For positive semi-definite matrix A and $p > 0$*

$$\mathrm{supp}(pA) = \mathrm{supp}(A).$$

For positive semi-definite matrices A, B, $A \leq pB$ for some $p > 0$ iff

$$\mathrm{supp}(A) \subseteq \mathrm{supp}(B).$$

Proof. Using the spectrum decomposition [43] of A, we know that for any $p > 0$,

$$\mathrm{supp}(pA) = \mathrm{supp}(A).$$

Now we only need to show

$$0 \leq A \leq B \Leftrightarrow \mathrm{supp}(A) \subseteq \mathrm{supp}(B).$$

Let P_A and P_B denote the projections on the support of A and B, respectively. According to Lemma A.1, we know that there exists $q_1, q_2 > 0$ such that

$$q_1 A \leq P_A,$$
$$P_B \leq q_2 B$$

On the other hand, if $\mathrm{supp}(A) \subseteq \mathrm{supp}(B)$, then $P_A \leq P_B$. Thus,

$$q_1 A \leq P_A \leq P_B \leq q_2 B \Longrightarrow A \leq \frac{q_2}{q_1} B.$$

□

Lemma A.3. *Suppose $A_{1,2}$ is a matrix on bipartite system $1, 2$, and Q_1 is a matrix on system 1, we have*

$$\mathrm{Tr}[A_{1,2}(Q_1 \otimes I_2)] = \mathrm{Tr}[(\mathrm{Tr}_2 A_{1,2})Q_1].$$

Proof. We can always write

$$A_{1,2} = \sum_{i,j} A_{1,2}^{(i,j)} \otimes |i\rangle\langle j|,$$

then

$$A_1 = \mathrm{Tr}_2 A_{1,2} = \sum_{i,j} A_{1,2}^{(i,j)} \otimes \langle i|j\rangle = \sum_i A_{1,2}^{(i,i)}.$$

On the other hand,

$$\begin{aligned} &\mathrm{Tr} && [A_{1,2}(Q_1 \otimes I_2)] \\ =\ &\mathrm{Tr} && [(\textstyle\sum_{i,j} A_{1,2}^{(i,j)} \otimes |i\rangle\langle j|)(Q_1 \otimes I_2)] \\ =\ &\mathrm{Tr} && [\textstyle\sum_{i,j} A_{1,2}^{(i,j)} Q_1 \otimes |i\rangle\langle j|] \\ =\ &\mathrm{Tr} && \textstyle\sum_i A_{1,2}^{(i,i)} Q = \mathrm{Tr}(P_1 Q_1). \end{aligned}$$

□

Lemma A.4. *For positive semi-definite matrix A, and projection P,* $\mathrm{supp}(A) \subseteq P$ *iff* $\mathrm{Tr}(PA) = \mathrm{Tr}A$.

Proof. Let the spectrum decomposition [43] of A is as

$$A = \sum_{i=0}^{m-1} \lambda_i |\psi_i\rangle\langle\psi_i|$$

where $\lambda_i > 0$ are the eigenvalues, and $|\psi_i\rangle$ is the corresponding eigenvector of λ_i.

For any pure state $|\psi\rangle$

$$\mathrm{Tr}(P|\psi\rangle\langle\psi|) \leq 1$$

and the equality is valid iff $|\psi\rangle \in P$.

Therefore,

$$\mathrm{Tr}(PA) = \sum_{i=0}^{m-1} \lambda_i \mathrm{Tr}(P|\psi_i\rangle\langle\psi_i|) = \sum_{i=0}^{m-1} \lambda_i$$

iff for all $0 \leq i \leq m-1$

$$|\psi_i\rangle \in P$$

The rest follows from the definition of support. □

Lemma 4.6. For positive semi-definite matrix A, $s \subseteq [n]$ and projection P_s on qubits s, $\mathrm{supp}(\mathrm{Tr}_s A) \subseteq P_s$, iff $\mathrm{supp}(A) \subseteq P_s \otimes I_s$.

Proof. According to Lemma A.3, $\mathrm{Tr}[A(P{-}S{\otimes}I_s)] = \mathrm{Tr}[(\mathrm{Tr}_s A)P_s]$. Therefore, if $\mathrm{supp}(\mathrm{Tr}_s A) \subseteq P_s$, we know that $\mathrm{Tr}[(\mathrm{Tr}_s A)P_s] = \mathrm{Tr}A = \mathrm{Tr}(\mathrm{Tr}_s A)$. If $\mathrm{supp}(A) \subseteq P_s \otimes I_s$, then $\mathrm{Tr}[A(P_s \otimes I_s)] = \mathrm{Tr}A$.

The rest is due to Lemma A.4. □

Lemma A.5. *If $P \subseteq Q$, then $\mathrm{supp}(\mathrm{Tr}_s P) \subseteq \mathrm{supp}(\mathrm{Tr}_s Q)$.*

Lemma A.6. $\mathrm{supp}(\mathrm{Tr}_{t\backslash s}[P_s \otimes I_{t\backslash s}] = P_s$.

Lemma A.7. *In an n-qubit system and for invertible matrices $A_0, A_1, \cdots, A_{n-1}$, and for any quantum states ρ and $\sigma = (A_0 \otimes A_1 \otimes \cdots \otimes A_{n-1})\rho(A_0 \otimes A_1 \otimes \cdots \otimes A_{n-1})^\dagger$, and for any $L \subseteq [n]$, we have*

$$\mathrm{supp}(\sigma_L) = \mathrm{supp}(\otimes_{i\in L} A_i \rho_L \otimes_{i\in L} A_i^\dagger).$$

Proof. We only prove it for $L = \{0,1\}$, the rest is similar.

Let $P_{0,1} = \mathrm{supp}(\rho_{0,1})$, we have

$$\begin{array}{ll} & \mathrm{supp}(\rho) \subseteq P_{0,1} \otimes I_{2,3,\cdots,n} \\ & \text{(Definition and Lemma 4.6)} \\ \Longrightarrow & \rho \leq P_{0,1} \otimes I_{2,3,\cdots,n} \\ & \text{(Lemma A.1)} \\ \Longrightarrow & (A_0 \otimes \cdots \otimes A_{n-1})\rho(A_0 \otimes \cdots \otimes A_{n-1})^\dagger \\ \leq & (A_0 \otimes \cdots \otimes A_{n-1})(P_{0,1} \otimes I_{2,3,\cdots,n})(A_0 \otimes \cdots \otimes A_{n-1})^\dagger \\ \Longrightarrow & \sigma \leq (A_0 \otimes A_1)P_{0,1}(A_0 \otimes A_1)^\dagger \otimes A_2 A_2^\dagger \otimes \cdots \otimes A_{n-1}A_{n-1}^\dagger \\ \Longrightarrow & \sigma_{0,1} \leq (A_0 \otimes A_1)P_{0,1}(A_0 \otimes A_1)^\dagger \mathrm{Tr}(A_2 A_2^\dagger) \cdots \mathrm{Tr}(A_{n-1}A_{n-1}^\dagger) \\ & \text{(Partial trace preserves } \leq) \\ \Longrightarrow & \sigma_{0,1} \leq p(A_0 \otimes A_1)P_{0,1}(A_0 \otimes A_1)^\dagger \text{ for some } p>0 \\ & \text{(Lemma A.2)} \\ \Longrightarrow & \sigma_{0,1} \leq q(A_0 \otimes A_1)\rho_{0,1}(A_0 \otimes A_1)^\dagger \text{ for some } q>0 \\ & (\mathrm{supp}(P_{0,1}) \subseteq \mathrm{supp}(\rho_{0,1}) \text{ and Lemma A.2}). \end{array}$$

On the other hand, there exists $r > 0$ such that

$$\rho_{0,1} \leq r(A_0^{-1} \otimes A_1^{-1})\sigma_{0,1}(A_0^{-1} \otimes A_1^{-1})^\dagger$$

by observing

$$\rho = (A_0^{-1} \otimes A_1^{-1} \otimes \cdots \otimes A_{n-1}^{-1})\sigma(A_0^{-1} \otimes A_1^{-1} \otimes \cdots \otimes A_{n-1}^{-1})^\dagger.$$

Therefore,

$$\mathrm{supp}(\sigma_L) = \mathrm{supp}(\otimes_{i\in L} A_i \rho_L \otimes_{i\in L} A_i^\dagger).$$

□

Lemma 5.1. Suppose $S \trianglelefteq T$. $\forall \mathcal{P}, \mathcal{Q} \in AbsDom(T)$: if $\mathcal{P} \sqsubseteq \mathcal{Q}$, then $\alpha_{T\to S}(\mathcal{P}) \sqsubseteq \alpha_{T\to S}(\mathcal{Q})$.

Proof. Suppose $\mathcal{P} = (P_{t_1}, \cdots, P_{t_m})$ and $\mathcal{Q} = (Q_{t_1}, \cdots, Q_{t_m})$. From $\mathcal{P} \sqsubseteq \mathcal{Q}$, we have $P_{t_i} \subseteq Q_{t_i}$ for all i. Additionally, suppose

$$\begin{aligned} \gamma_{S\to T}(P_{t_1}, \cdots, P_{t_m}) &= (R_{s_1}, \cdots, R_{s_m}) \\ \gamma_{S\to T}(Q_{t_1}, \cdots, Q_{t_m}) &= (V_{s_1}, \cdots, V_{s_m}). \end{aligned}$$

We have

$$\begin{aligned} R_{s_i} &= \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{Tr}_{t_j\backslash s_i} P_{t_j}) && \text{(by definition of } \alpha_{T\to S}) \\ &\subseteq \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{Tr}_{t_j\backslash s_i} Q_{t_j}) && \text{(by } P_{t_i} \subseteq Q_{t_i} \text{ and Lemma A.5)} \\ &= V_{s_i} \end{aligned}$$

Hence, $\alpha_{T\to S}(\mathcal{P}) \sqsubseteq \alpha_{T\to S}(\mathcal{Q})$. □

Lemma 5.2. Suppose $S \trianglelefteq T$. $\forall \mathcal{P}, \mathcal{Q} \in AbsDom(S)$: if $\mathcal{P} \sqsubseteq \mathcal{Q}$, then $\gamma_{S\to T}(\mathcal{P}) \sqsubseteq \gamma_{S\to T}(\mathcal{Q})$.

Proof. Suppose $\mathcal{P} = (P_{s_1}, \cdots, P_{s_m})$ and $\mathcal{Q} = (Q_{s_1}, \cdots, Q_{s_m})$. From $\mathcal{P} \sqsubseteq \mathcal{Q}$, we have $P_{s_i} \subseteq Q_{s_i}$ for all i. Additionally, suppose

$$\begin{aligned} \gamma_{S\to T}(P_{s_1}, \cdots, P_{s_m}) &= (R_{t_1}, \cdots, R_{t_m}) \\ \gamma_{S\to T}(Q_{s_1}, \cdots, Q_{s_m}) &= (V_{t_1}, \cdots, V_{t_m}). \end{aligned}$$

We have, for all j,

$$\begin{aligned} R_{t_j} &= \bigcap_{s_i : s_i \subseteq t_j} P_{s_i} \otimes I_{t_j\backslash s_i} && \text{(by definition of } \gamma_{S\to T}) \\ &\subseteq \bigcap_{s_i : s_i \subseteq t_j} Q_{s_i} \otimes I_{t_j\backslash s_i} && \text{(by } \mathcal{P} \sqsubseteq \mathcal{Q} \text{ and Lemma 4.1)} \\ &= V_{t_j} && \text{(by definition of } \gamma_{S\to T}) \end{aligned}$$

Hence, $\gamma_{S\to T}(\mathcal{P}) \sqsubseteq \gamma_{S\to T}(\mathcal{Q})$. □

Theorem 5.3. If $T \trianglelefteq R$ and $S \trianglelefteq T$, then $\alpha_{T\to S} \circ \alpha_{R\to T}(\mathcal{R}) \sqsubseteq \alpha_{R\to S}(\mathcal{R})$. If $R = [n]$, we get this special case with a stronger property: $\alpha_{T\to S} \circ \alpha_{[n]\to T} = \alpha_{[n]\to S}$.

Proof. Let

$$\begin{aligned} \mathcal{R} &= (V_{r_1}, \cdots, V_{r_m}) \\ \alpha_{R\to T}(\mathcal{R}) &= (Q_{t_1}, \cdots, Q_{t_m}) \\ Q_{t_j} &= \bigcap_{r_l : t_j \subseteq r_l} \mathrm{supp}(\mathrm{Tr}_{r_l\backslash t_j} V_{r_l}). \end{aligned}$$

Then, according to Lemma 4.3 and Lemma 4.4, we have

$$\alpha_{T\to S}(Q_{t_1}, \cdots, Q_{t_m}) = (P_{s_1}, \cdots, P_{s_m})$$

where

$$\begin{aligned} P_{s_i} &= \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{Tr}_{t_j\backslash s_i}\, Q_{t_j}) \\ &= \bigcap_{t_j : s_i \subseteq t_j} \mathrm{supp}(\mathrm{Tr}_{t_j\backslash s_i}\, (\bigcap_{r_l : t_j \subseteq r_l} \mathrm{supp}(\mathrm{Tr}_{r_l\backslash t_j} V_{r_l}))) \\ &\subseteq \bigcap_{t_j : s_i \subseteq t_j} (\bigcap_{r_l : t_j \subseteq r_l} \mathrm{supp}(\mathrm{Tr}_{t_j\backslash s_i} \mathrm{supp}(\mathrm{Tr}_{r_l\backslash t_j} P)) \\ &= \bigcap_{r_l : s_i \subseteq r_l} \mathrm{supp}(\mathrm{Tr}_{r_l\backslash s_i} V_{r_l})). \end{aligned}$$

By recalling the definition of $\alpha_{R\to S}$, it means,

$$\alpha_{T\to S} \circ \alpha_{R\to T}(\mathcal{R}) \sqsubseteq \alpha_{R\to S}(\mathcal{R}).$$

If $R = [n]$, the third line of the above proof becomes $=$ instead of $\subseteq$ because there is only one l, therefore in that case,

$$\alpha_{T\to S} \circ \alpha_{[n]\to T}(P) = \alpha_{[n]\to S}(P).$$

□

Theorem 5.4. If $T \trianglelefteq R$ and $S \trianglelefteq T$, then $\gamma_{T\to R} \circ \gamma_{S\to T} = \gamma_{S\to R}$.

Proof. Let

$$\gamma_{S\to T}(\mathcal{P}) = (Q_{t_1},\cdots,Q_{t_m})$$
$$Q_{t_j} = \bigcap_{s_i':s_i\subseteq t_j} P_{s_i}\otimes I_{t_j\setminus s_i}.$$

We have

$$\gamma_{T\to R}(Q_{t_1},\cdots,Q_{t_m}) = (V_{r_1},\cdots,V_{r_m})$$

where

$$\begin{aligned} V_{r_l} &= \bigcap_{t_j:t_j\subseteq r_l} Q_{t_j}\otimes I_{r_l\setminus t_j},\\ &= \bigcap_{t_j:t_j\subseteq r_l}\Big(\bigcap_{s_i:s_i\subseteq t_j} P_{s_i}\otimes I_{t_j\setminus s_i}\Big)\otimes I_{r_l\setminus t_j},\\ &= \bigcap_{t_j:t_j\subseteq r_l}\ \bigcap_{s_i:s_i\subseteq t_j}(P_{s_i}\otimes I_{t_j\setminus s_i})\otimes I_{r_l\setminus t_j},\\ &= \cap_{r_l:s_i\subseteq r_l}\ P_{s_i}\otimes I_{r_l\setminus s_i}.\end{aligned}$$

That proves

$$\gamma_{T\to R}\circ\gamma_{S\to T}(\mathcal{P}) = \gamma_{S\to R}(\mathcal{P}).$$

□

Lemma 7.1. For an assertion

$$A = \operatorname{span}\{|a_0a_1\cdots a_{n-1}\rangle, |b_0b_1\cdots b_{n-1}\rangle\},$$

we have $\operatorname{proj}(A) = \gamma_{S\to[n]}(\alpha_{[n]\to S}(\operatorname{proj}(A)))$ holds if S is "connected".

Proof. We only prove it for $S = (\{0,1\},\{0,2\},\cdots,\{n-1,n\})$. The general case follows from similar arguments. To compute $\alpha_{[n]\to S}(\operatorname{proj}(A)) = (P_{0,1},P_{0,2},\cdots,P_{n-2,n-1})$, we let

$$\begin{aligned}\rho =&0.5|a_0\rangle\langle a_0|\otimes|a_1\rangle\langle a_1|\cdots\otimes|a_{n-1}\rangle\langle a_{n-1}|\\ &+0.5|b_0\rangle\langle b_0|\otimes|b_1\rangle\langle b_1|\cdots\otimes|b_{n-1}\rangle\langle b_{n-1}|,\end{aligned}$$

We observe that there are $p,q>0$ such that

$$\begin{aligned}&\operatorname{supp}(\rho) = \operatorname{proj}(A)\\ \Longrightarrow&\operatorname{supp}(\rho)\subseteq\operatorname{proj}(A), \operatorname{proj}(A)\subseteq\operatorname{supp}(\rho)\\ \Longrightarrow&p\rho\le\operatorname{proj}(A)\le q\rho \qquad\text{(Lemma A.2)}.\end{aligned}$$

According to the fact that partial trace preserve the order of positive semi-definite matrices [43], we have

$$p\mathrm{Tr}_{[n]\setminus\{i,j\}}\rho\le\mathrm{Tr}_{[n]\setminus\{i,j\}}\operatorname{proj}(A)\le q\mathrm{Tr}_{[n]\setminus\{i,j\}}\rho.$$

According to

$$\rho_{i,j} = \mathrm{Tr}_{[n]\setminus\{i,j\}}\rho = 0.5(|a_i\rangle\langle a_i|\otimes|a_j\rangle\langle a_j| + |b_i\rangle\langle b_i|\otimes|b_j\rangle\langle b_j|),$$

Lemma A.2 and the definition of support, we have

$$P_{i,j} = \operatorname{supp}(\rho_{i,j}) = \operatorname{span}\{|a_i\rangle|a_j\rangle, |b_i\rangle|b_j\rangle\}.$$

Assume $|a_i\rangle$ and $|b_i\rangle$ are linear independent for all i. Then there exist invertible matrix A_i such that

$$A_i|a_i\rangle = |0\rangle,$$
$$A_i|b_i\rangle = |1\rangle.$$

Let

$$\begin{aligned}Q &= \operatorname{span}\{(A_0\otimes A_1\otimes\cdots\otimes A_{n-1})|\psi\rangle \mid |\psi\rangle\in\operatorname{proj}(A)\}\\ &= \operatorname{span}\{|0\rangle^{\otimes n}, |1\rangle^{\otimes n}\}\\ Q_{i,j} &= \operatorname{span}\{|00\rangle,|11\rangle\}.\end{aligned}$$

We have

$$\begin{aligned}&|\psi\rangle\in\gamma_{S\to[n]}(\alpha_{[n]\to S}(\operatorname{proj}(A)))\\ \Longleftrightarrow&|\psi\rangle\langle\psi|\subseteq P_{0,1},P_{0,2},\cdots,P_{n-2,n-1},\\ &|\phi\rangle = (A_0\otimes A_1\otimes\cdots\otimes A_{n-1})|\psi\rangle\\ \Longleftrightarrow&|\phi\rangle\langle\phi|\subseteq Q_{0,1},Q_{1,2},\cdots,Q_{n-2,n-1} \qquad\text{(Lemma A.7)}\\ \Longleftrightarrow&|\phi\rangle = Q_{i,i+1}|\phi\rangle\ \text{ for all } 0\le i<n-1 \text{ (Lemma A.1)}\\ \Longleftrightarrow&\cdots\\ \Longleftrightarrow&|\phi\rangle = Q_{n-2,n-1}\cdots Q_{1,2}Q_{0,1}|\phi\rangle\\ &=a_{0,0,\cdots,0}|0\rangle^{\otimes n} + a_{1,1,\cdots,1}|1\rangle^{\otimes n}\\ &\text{for some } a_{0,0,\cdots,0},a_{1,1,\cdots,1}\in\mathbb{C}\\ \Longleftrightarrow&|\phi\rangle\in Q = \operatorname{span}\{|0\rangle^{\otimes n},|1\rangle^{\otimes n}\}\\ \Longleftrightarrow&|\psi\rangle\in\operatorname{proj}(A) \qquad\text{(Lemma A.7)},\end{aligned}$$

where in the second and third lines, $\subseteq$ denotes the relation between projections, and $P_{i,j}$ and $Q_{i,j}$ are regarded as n-qubit projections $P_{i,j}\otimes I_{[n]\setminus\{i,j\}}$ and $Q_{i,j}\otimes I_{[n]\setminus\{i,j\}}$, respectively.

That is,

$$\operatorname{proj}(A) = \gamma_{S\to[n]}(\alpha_{[n]\to S}(\operatorname{proj}(A))).$$

If some $|a_i\rangle$ and $|b_i\rangle$ are linear dependent, we can choose invertible matrix A_i for such i such that

$$A_i|a_i\rangle = |0\rangle,$$
$$A_i|b_i\rangle = \lambda|0\rangle,$$

for some λ. We can have

$$Q = \operatorname{span}\{|0\rangle^{\otimes n}, |i_1,i_2,\cdots,i_n\rangle\}.$$

Let

$$s = \{j|\ 0\le j\le n-1,\ |a_j\rangle \text{ and } |b_j\rangle \text{ are linear dependent}\}.$$

Then

$$Q_{i,j} = \operatorname{span}\{|00\rangle,|11\rangle\}(\text{for } i,j\notin s,)$$
$$Q_{i,j} = \operatorname{span}\{|00\rangle\}(\text{for } i,j\in s).$$

We still have

$$\begin{aligned}&|\psi\rangle\langle\psi|\subseteq P_{0,1},P_{0,2},\cdots,P_{n-2,n-1}\\ \Longleftrightarrow&|\phi\rangle\langle\phi|\subseteq Q_{0,1},Q_{0,2},\cdots,Q_{n-2,n-1}. \qquad\text{(Lemma A.7)}\end{aligned}$$

Directly, we observe $|\phi\rangle$ can be written as $\otimes_{i\in s}|0\rangle_i\otimes|\tau\rangle_{[n]\setminus s}$. By employing the argument for linear independent $|a_j\rangle$ and $|b_j\rangle$, we know that

$$\begin{aligned}&|\tau\rangle_{[n]\setminus s}\in\operatorname{span}\{\otimes_{j\notin s}|0\rangle,\otimes_{j\notin s}|1\rangle\}\\ \Longleftrightarrow&|\phi\rangle\in Q,\\ \Longleftrightarrow&|\psi\rangle\in\operatorname{proj}(A) \quad\text{(Lemma A.7)}\end{aligned}$$

Therefore,

$$\operatorname{proj}(A) = \gamma_{S\to[n]}(\alpha_{[n]\to S}(\operatorname{proj}(A))).$$

□