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:

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|J [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 2n 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 2300 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.
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 $\{1, 2\}, \{2, 3\}, \ldots, \{19, 20\}$. 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 concretzing 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.
superposition is a maximally entangled state.

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.

The GHZ program creates a superposition of two states. 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 "endianess": the qubit registers 0, 1, 2 are ordered top to bottom in

In the diagram, each of the three horizontal rows depicts a qubit register that holds a single qubit. 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 assertion 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.

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. 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:

\[
\text{CNOT} = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
\end{pmatrix}
\]

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 assertion 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.

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.

Figure 1. The GHZ program for 3 qubits.
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 \(2^{300}\), 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 \(|00 \ldots 0\rangle\) and \(|11 \ldots 1\rangle\). 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 \([n] = \{0, 1, \ldots , n - 1\}\), 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 \(A^\dagger = (A^T)^\dagger\), where \(A^T\) is the transpose of \(A\), and \(B^\dagger\) 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, \(\text{Tr}(A) = \sum A_{ii}\). 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 \(||\psi|| = \sqrt{\langle \psi | \psi \rangle}\).

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 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, \(H|0\rangle = |+\rangle\) and \(H|1\rangle = |-\rangle\).

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 \(|00\rangle\) 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, \(||\psi|| = 1\). We can also represent \(|\psi\rangle\) by the density matrix \(\rho = |\psi\rangle \langle \psi|\). Every density matrix \(\rho\) is positive semi-definite and satisfies \(\text{Tr}(\rho) = 1\). 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 \times 8
matrix:
\[
\text{CCNOT} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

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 \( U_{F_i} \) means that the gate operates on the qubits in the list \( F_i \). 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 \( U_{F_i} \) appropriately. The idea is that \( U_{F_i} \) applies to the register(s) in \( F_i \) while leaving the rest of the registers untouched. Specifically, we can regard \( U_{F_i} \) as a unitary matrix that applies to the entire \( n \)-qubit register in the following way. For example, \( F_1 \) has size 1 and \( F_1 = \{i\} \), then we can view \( U_{F_1} \) 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,
\[
(\otimes_{k > i} I) \otimes U \otimes (\otimes_{0 \leq j < i} I) = U \otimes I_{n\setminus\{i\}},
\]
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 \( k_0, k_1, \ldots, k_{n-1} \in \{0, 1\}, \)
\[
U_{F_i}([k_{n-1} \cdots k_0]) = |k_{n-1} \cdots k_{i+1} \rangle \langle U_{[i]} | k_{i-1} \cdots k_0 \rangle,
\]
where the tensor products are implicit.

Similarly, a two-qubit unitary matrix \( U_{F_i} \) applied to registers \( F_i = \{i, j\} \) can be regarded as the following unitary matrix applied to an \( n \)-qubit register
\[
U_{F_i} \otimes I_{n\setminus\{i, j\}}.
\]
We say that this achieves an expansion of a unitary matrix via a tensor product. Notice that the formula uses \( U_{F_i} \) 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 \(|\psi\rangle \). Similarly, if we represent the quantum state as a density matrix \( \rho \) 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 \( C^{d_1}, C^{d_2} \) be the Hilbert spaces of two quantum systems considered in isolation. Then the composite system has a state space modeled by the tensor product \( C^{d_1} \otimes C^{d_2} \). The notion of partial trace is needed to extract the state of a subsystem. Formally, the partial trace over \( C^{d_1} \) is a mapping \( \text{Tr}_1(\cdot) \) from operators on \( C^{d_1} \otimes C^{d_2} \) to operators on \( C^{d_2} \) defined by the following equation: \( \text{Tr}_1(\langle \psi_1 | \otimes | \psi_2 \rangle \otimes \rho_{\psi_1} \langle \psi_1 | \otimes | \psi_2 \rangle) = \langle \psi_1_1 | \otimes | \psi_2 \rangle \langle \psi_1_1 | \otimes \rho_{\psi_1} \langle \psi_1_1 | \otimes | \psi_2 \rangle \) for all \( | \psi_1 \rangle, | \psi_2 \rangle \in C^{d_1} \) and \( \rho_{\psi_1} \) are \( C^{d_1} \). The partial density matrix \( \rho_{\psi_1} \) over \( C^{d_2} \) can be defined symmetrically. Suppose that we have a composite system of two subsystems with state spaces \( C^{d_1}, C^{d_2} \), respectively, and it is in density state \( \rho \). Then the states of the first and second subsystems can be described by \( \text{Tr}_2(\rho) \), \( \text{Tr}_1(\rho) \), 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
\[
|\Phi\rangle = \frac{1}{2}(|00\rangle(|00\rangle + |01\rangle|11\rangle + |11\rangle|00\rangle + |11\rangle|11\rangle)
\]
then the partial traces \( \text{Tr}_1(|\Phi\rangle\langle\Phi|) \) and \( \text{Tr}_2(|\Phi\rangle\langle\Phi|) \) 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
\[
\rho = \frac{1}{4}(|00\rangle\langle00| + |01\rangle\langle01| + |10\rangle\langle10| + |11\rangle\langle11|).
\]

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 \( \rho \) and any \( T \subseteq [n] \), such as \( \{1\} \) or \{2, 3\} (containing systems 2 and 3), the joint state of subsystems \( T \) is
\[
\rho_T = \text{Tr}_{[n]\setminus T}(\rho),
\]
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 throughout this paper. For simplicity, we will just call them projections.

orthogonal projection matrix

For a density matrix $\rho$, its positive semi-definite matrix

Subspace spanned by the eigenvectors of $A$

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

If $U$ applies to qubits $F \subseteq s$, then

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.

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

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 $\nu$ (a vector) as a projection $\nu \nu^\dagger$ (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 $S = (s_1, \ldots, s_m)$ with $s_i \subseteq [n]$, we can define an abstract domain $\text{AbsDom}(S)$ as follows

$$\text{AbsDom}(S) = \{ (P_{s_1}, \ldots, P_{s_m}) \mid P_{s_i} = P_{s_i}^1 = P_{s_i}^2 \in \mathbb{C}^{2^{|s_i|} \times 2^{|s_i|}} \}.$$  

The following lemmas describe relationships among these 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 $s_i, s_j, s_k \subseteq [n]$, 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 $\rho$ 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 $(P \subseteq Q_1 \land P \subseteq Q_2)$.

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 $\text{Tr}_{s_i}(\text{Tr}_{s_j}A) = \text{Tr}_{s_i}(\text{Tr}_{s_j}A) = \text{Tr}_{s_i \cup s_j}A$.

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.

The Logic of Quantum Mechanics

The following lemmas describe relationships among these 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 $s_i, s_j, s_k \subseteq [n]$, 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 $\rho$ 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 $(P \subseteq Q_1 \land P \subseteq Q_2)$.

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 $\text{Tr}_{s_i}(\text{Tr}_{s_j}A) = \text{Tr}_{s_i}(\text{Tr}_{s_j}A) = \text{Tr}_{s_i \cup s_j}A$.

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.

The first five lemmas can be proved by the definition of partial trace and support. We prove the last lemma in the supplementary material.
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 $AbstDom(S)$, we have

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

where (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 $(2^n)^m = 2^{mn}$ different abstract domains.

We define an ordering $\sqsubseteq$ on $AbstDom(S)$ as follows. Suppose $S = (s_1, \ldots, s_m)$. If $P, Q \in AbstDom(S)$, then

$P := (P_{s_1}, P_{s_2}, \ldots, P_{s_m})$

$Q := (Q_{s_1}, Q_{s_2}, \ldots, Q_{s_m})$

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

For $S = (s_1, \ldots, s_m)$ and $T = (t_1, \ldots, t_m)$, we define $S \sqsubseteq T$ (pronounced “$T$ is finer than $S$”) iff $\forall i : s_i \sqsubseteq t_i$.

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

$\alpha_{T \rightarrow S} : AbstDom(T) \rightarrow AbstDom(S)$

$\gamma_{S \rightarrow T} : AbstDom(S) \rightarrow AbstDom(T)$

$\alpha_{T \rightarrow S}(Q_{t_1}, \ldots, Q_{t_m}) = (P_{s_1}, \ldots, P_{s_m})$ where $P_{s_i} = \bigcap_{t_i \sqsubseteq s_i} \text{supp}(\text{Tr}_{t_i \rightarrow s_i} Q_{t_i})$

$\gamma_{S \rightarrow T}(P_{s_1}, \ldots, P_{s_m}) = (Q_{t_1}, \ldots, Q_{t_m})$ where $Q_{t_i} = \bigcap_{s_i \sqsubseteq t_i} P_{s_i} \otimes I_{t_i \setminus s_i}$

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

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

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

Note though that in some cases, the dimensions of those projections are the same. In more detail, $\gamma_{S \rightarrow T}$ constructs $Q_{t_i}$ by possibly expanding some projections into larger projections and then intersecting them. Dually, $\alpha_{T \rightarrow S}$ 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 $S = ([12], [4])$ with $\mathcal{R} = (R_{12, 3}, R_{12, 4})$ such that

$R_{12, 3} = (|\Phi\rangle\langle\Phi| + |01\rangle\langle01|) \otimes |00\rangle\langle00|$

$R_{12, 4} = (|000\rangle|000\rangle + |010\rangle|010\rangle)$

where $|\Phi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$ is a two-qubit maximally entangled state. We compute $\alpha_{T \rightarrow R}(\mathcal{R}) = (Q_{12}, Q_4)$, where

$Q_{12} = |01\rangle\langle01|$

$Q_4 = |00\rangle\langle00|$

Therefore,

$\alpha_{T \rightarrow S} \circ \alpha_{R \rightarrow T}(\mathcal{R}) = (|1\rangle|1\rangle, |00\rangle\langle00|)$

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

$P_2 = \text{supp}(\text{Tr}_{1}|\Phi\rangle\langle\Phi| + |01\rangle\langle01|) \cap \text{supp}(|00\rangle\langle00| + |11\rangle\langle11|)$

$P_4 = |01\rangle\langle01|$

Therefore in general

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

5.2 Properties

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

Lemma 5.1 (Abstraction is monotonic). Suppose $S \sqsubseteq T$. $\forall P, Q \in AbstDom(T)$: if $P \subseteq Q$, then $\alpha_{T \rightarrow S}(P) \subseteq \alpha_{T \rightarrow S}(Q)$.

Lemma 5.2 (Concretization is monotonic). Suppose $S \sqsubseteq T$. $\forall P, Q \in AbstDom(S)$: if $P \subseteq Q$, then $\gamma_{S \rightarrow T}(P) \subseteq \gamma_{S \rightarrow T}(Q)$.

Theorem 5.1 (Weak Galois connection). Suppose $S \sqsubseteq T$. $\forall P \in AbstDom(T):$ if $Q \in AbstDom(T)$, then $\alpha_{T \rightarrow S}(Q) \subseteq P$. Moreover, if $Q = \alpha_{n \rightarrow m}(\mathcal{R})$ for some $\mathcal{R} \in AbstDom([n]^m)$, $Q \subseteq \gamma_{S \rightarrow T}(P)$ if $\alpha_{T \rightarrow S}(Q) \subseteq P$. 
Proof. Suppose
\[ \mathcal{P} = (P_{S_1}, \ldots, P_{S_m}) \]
\[ \mathcal{Q} = (Q_{T_1}, \ldots, Q_{T_m}) \].

We have
\[ Q_{T_j} \subseteq \gamma_{S \rightarrow T}(\mathcal{P}) \]
\[ Q_{T_j} \subseteq \bigcap_{i_j \leq j} P_{S_i} \otimes I_{S_i} (\forall j) \]
\[ \supp(\text{Tr}_{S_j} \mathcal{Q}_{T_j}) \subseteq P_{S_j} (\forall j) \]
\[ \bigcap_{j: i_j \leq j} \supp(\text{Tr}_{S_j} \mathcal{Q}_{T_j}) \subseteq P_{S_j} (\forall j) \]
\[ \alpha_{T \rightarrow S}(\mathcal{Q}) \subseteq \mathcal{P} \).

In the second step, we use the definition of \( \gamma_{S \rightarrow T} \); 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 \( \alpha_{T \rightarrow S} \).

If \( \mathcal{Q} = \alpha_{[n]^{-m \rightarrow T}}(\mathcal{R}) \), we have that \( \mathcal{R} = (R, \ldots, R) \) is an \( m \)-tuple of a giant projection \( R \):
\[ Q_{T_j} = \supp(\text{Tr}_{S_j} R) \]

From \( Q \subseteq \gamma_{S \rightarrow T}(\mathcal{P}) \), we use the above argument up to the fourth step to have
\[ \supp(\text{Tr}_{S_j} \mathcal{Q}_{T_j}) \subseteq P_{S_j} (\forall j, s_j) \]
\[ \supp(\text{Tr}_{S_j} \supp(\text{Tr}_{S_j} \mathcal{Q}_{T_j})) \subseteq P_{S_j} (\forall j, s_j) \]
\[ \supp(\text{Tr}_{S_j} \mathcal{Q}_{T_j}) \subseteq P_{S_j} (\forall j) \]
\[ \bigcap_{j: i_j \leq j} \supp(\text{Tr}_{S_j} \mathcal{Q}_{T_j}) \subseteq P_{S_j} (\forall j) \]
\[ \alpha_{T \rightarrow S}(\mathcal{Q}) \subseteq \mathcal{P} \).

\[ \blacksquare \)

Theorem 5.2 (Galois connection). \( \forall \mathcal{P} \in \text{AbsDom}(S) \):
\[ \forall \mathcal{Q} \in \text{AbsDom}([n]^m) ; Q \subseteq \gamma_{S \rightarrow T}(\mathcal{P}) \text{ if } \alpha_{[n]^{-m \rightarrow T}}(\mathcal{Q}) \subseteq \mathcal{P} \).

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

6 Abstract Operations

6.1 Definitions

Consider \( S = (s_1, \ldots, s_m) \) where \( s_i \subseteq [n] \) and \( \text{AbsDom}(S) \). 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 \( U^F \):
\[ U^F : \text{AbsDom}(S) \rightarrow \text{AbsDom}(S) \]

The first step is to define a finer set \( T = (T_1, \ldots, T_m) \), where \( T_i = s_i \cup s_F \). We can check easily that \( S \subseteq T \). For an abstract state \( \mathcal{Q} = (Q_{T_1}, \ldots, Q_{T_m}) \in \text{AbsDom}(T) \), we define
\[ U^F(\mathcal{Q}) = (UQ_{T_1}U^+, \ldots, UQ_{T_m}U^+) \]
where \( U \) is regarded as a unitary matrix applied on \( T_i \) in each \( UQ_{T_i}U^+ \) since \( s_F \subseteq T_i \). This means that before multiplications, we expand \( U \) appropriately via tensor product.

Now we can define an abstract step:
\[ U^{\mathcal{Q}} = \alpha_{T \rightarrow S}(U^F(\mathcal{Q})) \]

We avoid picking \( T = [n]^m \) because the computation of \( \gamma_{S \rightarrow [n]^m} \) would cost exponential time and space, and so would \( U^{\mathcal{Q}} \) and \( \alpha_{[n]^{-m \rightarrow S}} \).
Our definition of $U^\sharp$ uses a kind of partial concretization that for each of the $m$ abstractions enriches $s_1$ to $s_1 \cup s_F$, 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 $t_i = s_1 \cup s_F$ can be modified in many ways that also make our results hold; we only require $s_1 \subseteq t_i$. Intuitively, our choice of $s_1 \cup s_F$ 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 $\text{AbsDom}(S)$.

6.2 Properties

In this paper, the quantum state $\rho$ is always an $n$-qubit projection.

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

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

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

$$Q = (Q_{t_1}, \ldots, Q_{t_m})$$

$$Q_{t_i} = \text{supp}(\text{Tr}[\rho_{n^i, t_i}])$$

We have

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

$$UQ_{t_1}U^\dagger = U\text{supp}(\text{Tr}[\rho_{n^i, t_i}])U^\dagger$$

$$= \text{supp}(\text{Tr}[\rho_{n^i, t_i}](U\rho U^\dagger))$$

(Lem. 4.2 + 4.5)

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

**Proof.**

$$\alpha_{[n]^m \rightarrow S}(\rho) \subseteq \mathcal{P}$$

$$\implies \alpha_{T \rightarrow S} \circ \alpha_{[n]^m \rightarrow T}(\rho) \subseteq \mathcal{P}$$

$$\implies \alpha_{[n]^m \rightarrow T}(\rho) \subseteq \mathcal{Y}_{S \rightarrow T}(\mathcal{P})$$

$$\implies U^{cg}(\alpha_{[n]^m \rightarrow T}(\rho)) \subseteq U^{cg} \circ \mathcal{Y}_{S \rightarrow T}(\mathcal{P})$$

$$\implies \alpha_{[n]^m \rightarrow S}(U\rho U^\dagger) \subseteq U^{cg} \circ \mathcal{Y}_{S \rightarrow T}(\mathcal{P})$$

$$\implies \alpha_{[n]^m \rightarrow S}(U\rho U^\dagger) \subseteq U^\sharp(\mathcal{P})$$

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

**Theorem 6.1 (Abstract operations are monotonic).**

$\forall \mathcal{P}_1, \mathcal{P}_2 \in \text{AbsDom}(S)$: if $\mathcal{P}_1 \subseteq \mathcal{P}_2$, then $U^\sharp(\mathcal{P}_1) \subseteq U^\sharp(\mathcal{P}_2)$.

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

From the discussion in Section 5, we know that $\alpha_{[n]^m \rightarrow S}(\rho) \subseteq \mathcal{P}$ 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

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

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 $\nu$, the final state of the abstract interpretation is $\mathcal{P} \in \text{AbsDom}(S)$, and we define $\rho = \nu U^\dagger$, then $\alpha_{[n]^m \rightarrow S}(\rho) \subseteq \mathcal{P}$.

**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 $\rho_i$ to denote the concrete state (as a density matrix) after $i$ steps, so we have $\rho_{i+1} = U_{i+1} U_i^\dagger \rho_i U_{i+1}^\dagger$. 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 $0 \leq i \leq z$, that $\alpha_{[n]^m \rightarrow S}(\rho_i) \subseteq \mathcal{P}_i$.

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

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

Now we calculate

$$\alpha_{[n]^m \rightarrow S}(\rho_{i+1}) = \alpha_{[n]^m \rightarrow S}(U_{i+1} \rho_i U_{i+1}^\dagger)$$

(lem. $\rho_{i+1}$)

$$\subseteq U_{F_{i+1}}^\sharp(\mathcal{P}_i)$$

(Lem. 6.2)

$$= \mathcal{P}_{i+1}$$

(def. $\mathcal{P}_{i+1}$)

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

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 x 4 matrix is a projection. Notice that applying $H(|0\rangle)$ 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)
We find two such sets, namely we take the first and the third matrix in the third row and which the corresponding entry in 

\[ H \]

does not change the entries, that is, obtaining a more compact matrix. For the mentioned example, a more compact form is 

\[ \langle \langle 00 \rangle | (10) + \langle 11 \rangle | (01) + \langle 10 \rangle | (11) \rangle + \langle 11 \rangle | (01) + \langle 10 \rangle | (11) \rangle \].

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 

\[ \langle \langle 00 \rangle | (10) + \langle 11 \rangle | (01) + \langle 10 \rangle | (11) \rangle + \langle 11 \rangle | (01) + \langle 10 \rangle | (11) \rangle \].

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 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., \( \langle \langle 00 \rangle | (00) + \langle 11 \rangle | (11) \rangle + \langle 10 \rangle | (10) \rangle \). 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 

\[ \langle \langle 00 \rangle | (10) + \langle 11 \rangle | (01) + \langle 10 \rangle | (11) \rangle + \langle 11 \rangle | (01) + \langle 10 \rangle | (11) \rangle \].

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.

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

<table>
<thead>
<tr>
<th>Initial state:</th>
<th>(00)(00)</th>
<th>(00)(00)</th>
<th>(00)(00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply ( H^x(0) ):</td>
<td><a href="0+">0+</a></td>
<td><a href="0+">0+</a></td>
<td><a href="00">00</a></td>
</tr>
<tr>
<td>Apply ( H^y(1) ):</td>
<td><a href="+i">+i</a></td>
<td><a href="0+">0+</a></td>
<td><a href="00">00</a></td>
</tr>
<tr>
<td>Apply ( H^z(2) ):</td>
<td><a href="+i">+i</a></td>
<td><a href="1+">1+</a></td>
<td><a href="11">11</a></td>
</tr>
<tr>
<td>Apply ( \text{CNOT}^z(1,2) ):</td>
<td><a href="0+">0+</a> + <a href="1+">1+</a></td>
<td><a href="0+">0+</a> + <a href="1+">1+</a></td>
<td>( \frac{1}{2} \langle (01)</td>
</tr>
<tr>
<td>Apply ( \text{CNOT}^z(0,2) ):</td>
<td><a href="+i">+i</a> + <a href="+i">+i</a></td>
<td><a href="+i">+i</a> + <a href="+i">+i</a></td>
<td><a href="+i">+i</a></td>
</tr>
<tr>
<td>Apply ( H^x(0) ):</td>
<td><a href="0+">0+</a> + <a href="1+">+i</a></td>
<td><a href="0+">0+</a> + <a href="1+">+i</a></td>
<td><a href="0+">0+</a></td>
</tr>
<tr>
<td>Apply ( H^y(1) ):</td>
<td><a href="00">00</a> + <a href="11">11</a></td>
<td><a href="00">00</a> + <a href="11">11</a></td>
<td><a href="00">00</a> + <a href="11">11</a></td>
</tr>
<tr>
<td>Apply ( H^z(2) ):</td>
<td><a href="00">00</a> + <a href="11">11</a></td>
<td><a href="00">00</a> + <a href="11">11</a></td>
<td><a href="00">00</a> + <a href="11">11</a></td>
</tr>
</tbody>
</table>

### Figure 3. The application of \( H^z(0) \) to the initial state.

<table>
<thead>
<tr>
<th>Initial state:</th>
<th>(00)(00)</th>
<th>(00)(00)</th>
<th>(00)(00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply ( \gamma_{S\rightarrow T} ):</td>
<td>(00)(00)</td>
<td>(00)(00)</td>
<td>(00)(00)</td>
</tr>
<tr>
<td>Apply ( H^y^9_{01} ):</td>
<td><a href="0+">0+</a></td>
<td><a href="0+">0+</a></td>
<td><a href="0+">0+</a></td>
</tr>
<tr>
<td>Apply ( \alpha_{T\rightarrow S} ):</td>
<td><a href="0+">0+</a></td>
<td><a href="0+">0+</a></td>
<td><a href="0+">0+</a></td>
</tr>
</tbody>
</table>

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 \( F = \{(0,0)\} \), so we get \( T = \{(0,1) \cup (0,2) \cup (1,2) \cup (0)\} = \{(0,1), (0,2), (1,2)\} \). Figure 3 shows the initial state, the tuple of matrices after applying \( \gamma_{S\rightarrow T} \), then after applying \( H_{01}^y \), and finally after applying \( \alpha_{T\rightarrow S} \). Notice that the final tuple of matrices in Figure 3 matches the tuple of matrices in Figure 2 after we apply \( H^z(0) \). Notice also that two of the matrices are of size \( 8 \times 8 \), 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 \( \gamma_{S\rightarrow T} \). The definition of \( \gamma_{S\rightarrow T} \) 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 \( \gamma_{S\rightarrow T} \), we see that \( Q_{0,1} = P_{0,1} \).

Next we have \( U^g_{0} \), and here \( U \) is \( H \). We need to expand \( H \) to work with a \( 4 \times 4 \) 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 \( \alpha_{T\rightarrow S} \). The definition of \( \alpha_{T\rightarrow S} \) 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 \( \alpha_{T\rightarrow S} \). This computation intersects two matrices that each is obtained by

\begin{align*}
\text{Tr} & \left( H^z(0) \right) = \\
& = (00)(00) + (00)(00) + (00)(00) \\
& = (00)(00) + (00)(00) + (00)(00).
\end{align*}
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 \(j\) that are more than the number of atoms in the known universe. 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 \(U^g\). The reason is that the step of applying \(\alpha_{T-S}\) is faster, while the step of applying \(y_{S-T}\) is within a factor of 2 slower.

The main work of applying \(U^g\) 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. These 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:

\[
A = \text{span}([a_1] [a_2] \cdots [a_n], [b_1] [b_2] \cdots [b_n])
\]

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

\[
\text{proj}(A)[a_1][a_2] \cdots [a_n] = [a_1][a_2] \cdots [a_n],
\]

\[
\text{proj}(A)[b_1][b_2] \cdots [b_n] = [b_1][b_2] \cdots [b_n].
\]

Using an appropriately chosen \(S\) and \(P \in \text{AbsDom}(S)\), we check \(P \subseteq \alpha_{[n] \rightarrow S}(\text{proj}(A))\). 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 \(S\). \(S = (s_1, \ldots, s_m)\) with \(s_i \subseteq [n]\) is called “connected” if for any \(i, j \in [n]\), there exists integer \(t\) and a sequence \(a_0, \ldots, a_t \in [n]^t\) such that \(a_0 = k, a_t = l\) and for any \(0 \leq i < t\), both \(a_i, a_{i+1} \in s_i\) for some \(1 \leq r_i < m\).

**Lemma 7.1.** For an assertion

\[
A = \text{span}([a_0][a_1] \cdots [a_{n-1}], [b_0][b_1] \cdots [b_{n-1}]),
\]

we have \(\text{proj}(A) = y_{S-[n]}(\alpha_{[n] \rightarrow S}(\text{proj}(A)))\) 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

\[
A_{GHZ_3} = \text{span}( |000\rangle, |111\rangle)
\]

In Section 5.3, we picked \(S = \{[0, 1], [0, 2], [1, 2]\}\). Now we can find \(\alpha_{[3] \rightarrow S}(\text{proj}(A_{GHZ_3}))\) equals

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

We see that \(\alpha_{[3] \rightarrow S}(\text{proj}(A_{GHZ_3}))\) is equal to the final abstract state \(P\) in Figure 2, that is, we have \(\alpha_{[3] \rightarrow S}(\text{proj}(A_{GHZ_3})) = P\), so \(P \subseteq \alpha_{[3] \rightarrow S}(\text{proj}(A_{GHZ_3}))\).

### 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 \(P \in \text{AbsDom}(S)\), and \(P \subseteq \alpha_{[n] \rightarrow S}(\text{proj}(A))\), then \(v \in A\).

**Proof.** Define \(\rho = vv^\dagger\). We reason as follows:

\[
\alpha_{[n] \rightarrow S}(\rho) \subseteq P \quad \text{(Thm 6.2)}
\]

iff

\[
\rho \subseteq y_{S-[n]}(P) \quad \text{(Thm 5.2)}
\]

\[
\rho \subseteq y_{S-[n]}(\alpha_{[n] \rightarrow S}(\text{proj}(A)))
\]

\[
= \text{proj}(A) \quad \text{(Lem 7.1)}
\]

where in the third line, we use \(P \subseteq \alpha_{[n] \rightarrow S}(\text{proj}(A))\) and Lemma 5.2.

Next, notice that \(\rho(v) = (vv^\dagger)v = v(v^\dagger v) = v1 = v\). From \(\rho \subseteq \text{proj}(A)\) and \(\rho(v) = v\), we have \(\text{proj}(A)(v) = v\), 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
we check that every matrix is a nonzero projection.

ward, and we compute the support of a matrix using the
frequency and 16 GB main memory.

an Intel Core i7 processor with 2.2 GHz processor clock fre-

an automatically generated parser. Our implementations of
in 2,350 lines of Java, on top of an existing matrix library and

ation supports our claims.

In the remainder of this section, we show how our evalua-
tion 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 fre-
cquency and 16 GB main memory.

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 an-
swers three questions about our approach to quantum ab-
stract 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.

Benchmarks. We have a benchmark suite of 16 quantum
programs that can be divided into three families, see Fig-
ure 4. The column labeled $n$ shows the number of qubits.
Each program uses between 50 and 300 qubits and has be-
tween 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
$f : \{0, 1\}^n \rightarrow \{0, 1\}$, where $f(x) = a \cdot x + b$. 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

$$\text{span} \{ |00\rangle \}$$

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, includ-
ing quantum teleportation, which in turn enables the quan-
tum 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 al-
gorithm [33] for 63, 127, 255, and 300 qubits. The input to
Grover’s algorithm is a black-box predicate $f$ that has a do-
main 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

| 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 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 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 \( \text{AbsDom}(S) \), 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:
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: \(\alpha_{T \rightarrow S} \circ U_{F}^{\alpha} \circ Y_{S \rightarrow T}\). Let us use the terminology that the application of \(\alpha_{T \rightarrow S}\) is the alpha step, while the application of \(U_{F}^{\alpha}\) is the transformation step and the application of \(Y_{S \rightarrow T}\) 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 is that Grover step uses \(k = 5\) so all the matrices are of size \(256 \times 256\). 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 \(\alpha\) finds little work to do. Specifically, for any element \(s\) 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.

**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


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 $\text{supp}(A) = Q$, then there exists $r_1, r_2 > 0$ such that

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

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

$$\rho \leq r P.$$

**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 $\lambda_i$. Therefore,

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

Write it in the projection form, it is

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

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

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

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

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

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

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

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

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

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

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

Now we only need to show

$$0 \leq A \leq B \iff \text{supp}(A) \subseteq \text{supp}(B).$$

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

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

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

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

**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

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

**Proof.** We can always write

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

then

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

On the other hand,

$$\text{Tr} \left[ \left[ \sum_{i,j} A_{1,2}^{(i,j)} \otimes | i \rangle \langle j | \right] (Q_1 \otimes I_2) \right]$$

$$= \text{Tr} \left[ \left[ \sum_{i,j} A_{1,2}^{(i,j)} Q_1 \otimes | i \rangle \langle j | \right] \right]$$

$$= \text{Tr} \sum_i A_{1,2}^{(i,i)} Q = \text{Tr}(P_1 Q_1).$$

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

**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 $\lambda_i$.

For any pure state $| \psi \rangle$

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

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

Therefore,

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

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 $A$, $s \leq [n]$ and projection $P_s$ on qubits $s$, $\text{supp}(Tr_s(A)) \subseteq P_s$, if $\text{supp}(A) \subseteq P_s \otimes I_s$.

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

The rest is due to Lemma A.4.

Lemma A.5. If $P \subseteq Q$, then $\text{supp}(Tr_s P) \subseteq \text{supp}(Tr_s Q)$.

Lemma A.6. $\text{supp}(Tr_{j\setminus i}[P_s \otimes I_{j\setminus i}]) = P_s$.

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

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

Proof. We only prove it for $L = [0, 1]$, the rest is similar. Let $P_{0,1} = \text{supp}(\rho_{0,1})$, we have

$$\text{supp}(\rho) \subseteq P_{0,1} \otimes I_{2,3, \ldots, n}$$

(Definition and Lemma 4.6)

$$\Rightarrow \rho \leq P_{0,1} \otimes I_{2,3, \ldots, n}$$

(Lemma A.1)

$$\Rightarrow (A_0 \otimes \cdots \otimes A_{n-1})^\dagger \rho (A_0 \otimes \cdots \otimes A_{n-1})^\dagger \leq (A_0 \otimes \cdots \otimes A_{n-1})^\dagger P_{0,1} \otimes I_{2,3, \ldots, n} (A_0 \otimes \cdots \otimes A_{n-1})^\dagger$$

(Lemma A.2)

$$\Rightarrow \sigma \leq (A_0 \otimes A_1) P_{0,1} (A_0 \otimes A_1)^\dagger \otimes A_2^\dagger A_2 \otimes \cdots \otimes A_{n-1}^\dagger A_{n-1}$$

(Partial trace preserves $\leq$)

$$\Rightarrow \sigma_{0,1} \leq (A_0 \otimes A_1) P_{0,1} (A_0 \otimes A_1)^\dagger$$

(by some $p > 0$)

$$\Rightarrow \sigma_{0,1} \leq q (A_0 \otimes A_1) P_{0,1} (A_0 \otimes A_1)^\dagger$$

(by supp($P_{0,1}$) $\subseteq \text{supp}(\rho_{0,1}$) and Lemma A.2).

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

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

by observing

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

Therefore,

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

We have

$$R_s = \bigcap_{t_j : s_j \subseteq s_j} \text{supp}(Tr_{t_j \setminus s_j} P_{t_j})$$

(by definition of $\alpha_{T \setminus s}$)

$$\subseteq \bigcap_{t_j : s_j \subseteq s_j} \text{supp}(Tr_{t_j \setminus s_j} Q_{t_j})$$

(by $P_{t_j} \subseteq Q_{t_j}$ and Lemma A.5)

$$= V_{s_j}$$

Hence, $\alpha_{T \setminus s}(P) \subseteq \alpha_{T \setminus s}(Q)$.

Lemma 5.2. Suppose $S \subseteq T$. Suppose $\mathcal{P}, \mathcal{Q} \in \text{AbsDom}(S)$: if $\mathcal{P} \nsubseteq \mathcal{Q}$, then $\mathcal{Y}_{S \setminus T}(\mathcal{P}) \nsubseteq \mathcal{Y}_{S \setminus T}(\mathcal{Q})$.

Proof. Suppose $\mathcal{P} = (P_{s_1}, \ldots, P_{s_m})$ and $\mathcal{Q} = (Q_{t_1}, \ldots, Q_{t_m})$. From $\mathcal{P} \nsubseteq \mathcal{Q}$, we have $P_{s_i} \subseteq Q_{t_i}$ for all $i$. Additionally, suppose

$$\mathcal{Y}_{S \setminus T}(P_{s_1}, \ldots, P_{s_m}) = (R_{s_1}, \ldots, R_{s_m})$$

$$\mathcal{Y}_{S \setminus T}(Q_{t_1}, \ldots, Q_{t_m}) = (V_{t_1}, \ldots, V_{t_m}).$$

We have

$$R_{s_i} = \bigcap_{t_j : s_i \subseteq s_j} \text{supp}(Tr_{t_j \setminus s_j} P_{t_j})$$

(by definition of $\alpha_{T \setminus s}$)

$$\subseteq \bigcap_{t_j : s_i \subseteq s_j} \text{supp}(Tr_{t_j \setminus s_j} Q_{t_j})$$

(by $P_{t_j} \subseteq Q_{t_j}$ and Lemma A.5)

$$= V_{s_j}$$

Hence, $\alpha_{T \setminus s}(\mathcal{P}) \subseteq \alpha_{T \setminus s}(\mathcal{Q})$.

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

Proof. Let

$$\mathcal{R} = (V_{r_1}, \ldots, V_{r_m})$$

$$\alpha_{T \setminus S}(\mathcal{R}) = (Q_{t_1}, \ldots, Q_{t_m})$$

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

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

where

$$P_{s_i} = \bigcap_{t_j : s_i \subseteq s_j} \text{supp}(Tr_{t_j \setminus s_j} Q_{t_j})$$

$$= \bigcap_{t_j : s_i \subseteq s_j} \text{supp}(Tr_{t_j \setminus s_j} (\bigcap_{r_j : t_j \subseteq r_j} \text{supp}(Tr_{r_j \setminus t_j} V_{r_j})))$$

$$\subseteq \bigcap_{t_j : s_i \subseteq s_j} \text{supp}(Tr_{t_j \setminus s_j} \text{supp}(Tr_{r_j \setminus t_j} P_{r_j}))$$

$$= \bigcap_{r_j : s_i \subseteq r_i} \text{supp}(Tr_{r_j \setminus s_i} V_{r_i}).$$

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

$$\alpha_{T \setminus S} \circ \alpha_{R \setminus T}(\mathcal{R}) \subseteq \alpha_{R \setminus S}(\mathcal{R})$$

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

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

Theorem 5.4. If $T \subseteq R$ and $S \subseteq T$, then $\gamma_{T \setminus R} \circ \mathcal{Y}_{S \setminus T} = \mathcal{Y}_{S \setminus R}$. 

\[\]
Proof. Let
\[ Y_{S \rightarrow T}(P) = (Q_1, \ldots, Q_m) \]
\[ Q_{t_j} = \bigcap_{s_j: s_j \leq t_j} P_{s_j} \otimes I_{t_j \setminus s_j}. \]
We have
\[ Y_{T \rightarrow R}(Q_1, \ldots, Q_m) = (V_1, \ldots, V_m) \]
where
\[ V_{t_j} = \bigcap_{s_j: s_j \leq t_j} Q_{t_j} \otimes I_{r_j \setminus t_j}, \]
\[ = \bigcap_{s_j: s_j \leq t_j} \left( \bigcap_{s_j: s_j \leq t_j} P_{s_j} \otimes I_{t_j \setminus s_j} \otimes I_{r_j \setminus t_j} \right), \]
\[ = \bigcap_{s_j: s_j \leq t_j} P_{s_j} \otimes I_{t_j \setminus s_j} \otimes I_{r_j \setminus t_j}, \]
\[ = \bigcap_{s_j: s_j \leq t_j} P_{s_j} \otimes I_{t_j \setminus s_j}. \]
That proves
\[ Y_{T \rightarrow R} \circ Y_{S \rightarrow T}(P) = Y_{S \rightarrow R}(P). \]
\[ \square \]

**Lemma 7.1.** For an assertion
\[ A = \text{span}([a_0 a_1 \cdots a_{n-1}, |b_0 b_1 \cdots b_{n-1}\rangle], \]
we have \( \text{proj}(A) = Y_{S \rightarrow [n]}(\alpha_{[n]} \rightarrow S(\text{proj}(A))) \) holds if \( S \) is “connected”.

**Proof.** We only prove it for \( S = \{0, 1\}, \{0, 2\}, \ldots, \{n-1\} \). The general case follows from similar arguments. To compute \( \alpha_{[n]} \rightarrow S(\text{proj}(A)) = (P_{0,1}, P_{0,2}, \ldots, P_{n-2, n-1}) \), we let
\[ \rho = 0.5|a_0 \rangle \langle a_0| \otimes |a_1 \rangle \langle a_1| \cdots \otimes |a_{n-1} \rangle \langle a_{n-1}| \]
\[ + 0.5|b_0 \rangle \langle b_0| \otimes |b_1 \rangle \langle b_1| \cdots \otimes |b_{n-1} \rangle \langle b_{n-1}|. \]
We observe that there are \( p, q > 0 \) such that
\[ \text{supp}(\rho) = \text{proj}(A) \]
\[ \implies \text{supp}(\rho) \subseteq \text{proj}(A), \text{proj}(A) \subseteq \text{supp}(\rho) \]
\[ \implies \rho \leq \text{proj}(A) \leq q \rho \quad (\text{Lemma A.2}). \]
According to the fact that partial trace preserve the order of positive semi-definite matrices [43], we have
\[ p \text{Tr}_{[n] \setminus \{i,j\}} \rho \leq \text{Tr}_{[n] \setminus \{i,j\}} \text{proj}(A) \leq q \text{Tr}_{[n] \setminus \{i,j\}} \rho. \]
According to
\[ \rho_{i,j} = \text{Tr}_{[n] \setminus \{i,j\}} \rho = 0.5(|a_i \rangle \langle a_i| \otimes |a_j \rangle \langle a_j| + |b_i \rangle \langle b_i| \otimes |b_j \rangle \langle b_j|), \]
Lemma A.2 and the definition of support, we have
\[ P_{i,j} = \text{supp}(\rho_{i,j}) = \text{span}([a_i \langle a_j|, |b_i \rangle \langle b_j|]). \]
Assume \( |a_i \rangle \) and \( |b_j \rangle \) are linear independent for all \( i \). Then there exist invertible matrix \( A_i \) such that
\[ A_i |a_i \rangle = |0\rangle, \]
\[ A_i |b_j \rangle = |1\rangle. \]
Let
\[ Q = \text{span}((A_0 \otimes A_1 \otimes \cdots \otimes A_{n-1})|\psi\rangle |\psi\rangle \in \text{proj}(A)) \]
\[ = \text{span}(|0\rangle ^{\otimes n}, |1\rangle ^{\otimes n}) \]
\[ Q_{i,j} = \text{span}([00\rangle, [11\rangle]. \]
We have
\[ |\psi\rangle \in Y_{S \rightarrow [n]}(\alpha_{[n]} \rightarrow S(\text{proj}(A))) \]
\[ \iff |\psi\rangle \langle \psi | \subseteq P_{0,1}, P_{0,2}, \ldots, P_{n-2, n-1}, \]
\[ |\phi\rangle = (A_0 \otimes A_1 \otimes \cdots \otimes A_{n-1})|\phi\rangle \]
\[ \iff |\phi\rangle \langle \phi | \subseteq Q_{0,1}, Q_{1,2}, \ldots, Q_{n-2, n-1} \quad (\text{Lemma A.7}) \]
\[ \iff |\phi\rangle = Q_{i,i+1}|\phi\rangle \quad \forall 0 \leq i < n - 1 \quad (\text{Lemma A.1}) \]
\[ \iff \cdots \]
\[ \iff \iff |\phi\rangle \in Q = \text{span}(|0\rangle ^{\otimes n}, |1\rangle ^{\otimes n}) \]
\[ \iff |\phi\rangle \in \text{proj}(A) \quad (\text{Lemma A.7}), \]
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,
\[ \text{proj}(A) = Y_{S \rightarrow [n]}(\alpha_{[n]} \rightarrow S(\text{proj}(A))). \]
If some \( |a_i \rangle \) and \( |b_j \rangle \) are linear dependent, we can choose invertible matrix \( A_i \) for such \( i \) such that
\[ A_i |a_i \rangle = |0\rangle, \]
\[ A_i |b_j \rangle = |\lambda \rangle, \]
for some \( \lambda \). We can have
\[ Q = \text{span}(|0\rangle ^{\otimes n}, |1\rangle ^{\otimes n}). \]
We have
\[ |\psi\rangle \leq P_{0,1}, P_{0,2}, \ldots, P_{n-2, n-1} \]
\[ \iff |\phi\rangle \leq Q_{0,1}, Q_{0,2}, \ldots, Q_{n-2, n-1} \quad (\text{Lemma A.7}) \]
Directly, we observe \( |\phi\rangle \) can be written as \( \otimes_{i \in \{0\}} |r_{[n]}\rangle |s\rangle \).
By employing the argument for linear independent \( |a_i \rangle \) and \( |b_j \rangle \), we know that
\[ |r_{[n]}\rangle |s\rangle \in \text{span}(|0\rangle \otimes |0\rangle, |1\rangle \otimes |1\rangle \}
\[ \iff |\phi\rangle \in Q, \]
\[ \iff |\psi\rangle \in \text{proj}(A) \quad (\text{Lemma A.7}) \]
Therefore,
\[ \text{proj}(A) = Y_{S \rightarrow [n]}(\alpha_{[n]} \rightarrow S(\text{proj}(A))). \]
\[ \square \]