A Survey Towards the Verification of Quantum Parallel Programs

Yusuf Alnawakhtha Corbin McNeill

1 Introduction

One of the biggest challenges that quantum computing faces is the decoherence of qubits, which makes decreasing the operating time of a quantum algorithm just as essential for correctness as it is for efficiency. Additionally, due to the unavailability of quantum computers with large qubit capacity, being able to execute a program across multiple quantum computers would allow us to attain a large-capacity system [1]. Hence, distributed and parallel quantum programming are important paradigms of quantum computation.

As distributed and parallel quantum computation become more common, formal verification methods for distributed parallel quantum programming will become more valuable. Methods for verification currently exist for quantum while programs; however, quantum while programs represent only a subset of quantum programs which someone may wish to execute. Quantum computation in general is inherently distributed due to possible entanglement between registers, thus a more global structure is required in general [2].

In this paper, we provide a survey of the quantum Floyd-Hoare logic proposed by Mingsheng Ying [3] and discuss how future work may go about combining that with existing verification models for classical parallel programming. Namely, we will be referring to rules used for checking if two concurrent programs do not interfere with each other. Such an approach does not provide a complete model of verification, even in the classical case, but it might be a promising approach for a sound model of verification for quantum computation.

2 Preliminaries

We will use the quantum density operator model for representing quantum states of computation. States are represented by density operators, positive operators with trace equal to 1. Evolution of quantum states are represented by left-operating unitary operators. Furthermore, we will often refer to partial density operators, positive operators with trace no greater than one. We refer to set off all partial operators on $\mathcal{H}$ as $\mathcal{D}^{-}(\mathcal{H})$. Measurements are defined in terms of a set of possible measurement outcomes $M = \{M_m\}$. The probability of measuring an outcome of m on state ρ is $p_m = tr(M_m \rho M_m^\dagger)$. After observing a measurement of M_m, we are left with a new state of $\frac{M_m \rho M_m^\dagger}{p_m}$.

3 Quantum Programming Languages

3.1 While Programs

Current literature on Floyd-Hoare logic on quantum computers largely uses a quantum version of while programming language. We can define this while programming language via the following grammar from [3]:

$$S ::= \textbf{skip} \mid \bar{q} := 0 \mid \bar{q} := U\bar{q} \mid S_1; S_2 \mid \textbf{mesasure}\ M[\bar{q}] : \bar{S} \mid \textbf{while}\ M[\bar{q}] = 1\ \textbf{do}\ S$$

Intuitively, this language allows us to set and change variables, concatenate programs, branch our program conditional on the measurement of variable, and loop on the measurement of a variable, respectively. One think worth pointing out is the difference between the quantum measurement that occurs in the **measure** and the **while** statements. In the **measure**, the measurement $M[\bar{q}]$ can resolve to any number of measurements. $\bar{S}$ corresponds to as many programs as there are measurements and on any given measurement outcome runs the subprogram associated with that output.

In the **while** program, on the other hand, can only have two measurement outcomes, termed 0 and 1. The reason for this is that the **while** command only has two outputs. If the measurement of $M[\bar{q}]$ is 0, the command outputs the **skip** program. If the measurement is 1, the command outputs S. This notion of outputting will be formalized in Section 3.3.

3.2 Variables

As we saw while examining the while programming language, variables played a crucial role in defining the language. Here we will present two different notions of the variable space of a program as well as formally define the relationship between quantum variables and state spaces.

3.2.1 Purely Quantum Variables

The primary notion of variables in a while program for formal verification is defined by Ying in [3]. In this definition all the variables are quantum and can be either booleans or integers. Booleans are associated with the Hilbert space $\mathcal{H}_2$ and integers with the Hilbert space $\mathcal{H}_\infty$. We refer to these Hilbert spaces as the associated state spaces for these quantum variables. For the variable q we will refer to its associated Hilbert space as $\mathcal{H}_q$.

As we seek to generalize the notion of state space from a single variable to an entire program, it will first be necessary to form a well defined notion of what the variable space associated a program S is. Ying presents the following recursive definition:

1. $\text{Var}(\textbf{skip}) := \emptyset$
2. $\text{Var}(q = 0) := \{q\}$
3. $\text{Var}(\bar{q} = U\bar{q}) := \{\bar{q}\}$

4. $\text{Var}(S_1; S_2) = \text{Var}(S_1) \cup \text{Var}(S_2)$
5. $\text{Var}(\textbf{measure } M[\bar{q}] : \bar{S}) := \{\bar{q}\} \cup \left(\bigcup_{S_m \in \bar{S}} \text{Var}(S_m)\right)$
6. $\text{Var}(\textbf{while } M[\bar{q}] = 1 \textbf{ do } S) := \{\bar{q}\} \cup \text{Var}(S)$

Now that we have defined a notion of the variable space associated with a program S, we can relate this variable set to a state space associated with S. We simply allow the state space of S to be the tensor product of all the variables associated with S. Formally, the state space of S is

$$\mathcal{H}_{\text{Var}(S)} := \bigotimes_{q \in \text{Var}(S)} \mathcal{H}_q.$$

This notion of state space will be necessary as we define transition rules from quantum while programs.

3.2.2 Mixed Quantum and Classical Variables

Chadha *et al.* [4] present an alternative notion of the variable space for a quantum program which we will describe here.

The first major difference between the system presented by Ying and the system presented by Chadha *et al.* is that Chadha's system allows for both quantum and classical variables instead of quantum variables only. Since quantum boolean and integral variables are simply a generalization of their classical counterparts, this addition does not make the mixed system intrinsically more powerful. It does, however, add an abstraction that some may find convenient when reasoning about the semantics of qunatum programs.

Additionally, [4] makes a modification to the integer data type. The integer data type presented in [3] was defined over $\mathcal{H}_\infty$ meaning that in order to represent a fixed sized integer, an infinite number of qubits were required. Chadha presents a much more practical notion of the integer using *integral registers* that have a bounded size. Both the classical and quantum variants of the integral registers can take values from the set $\{0, 1, \ldots, N-1\}$ when measured. This definition of integers allows for the system in [4] to operate with finite memory.

3.3 Transition Rules

Semantically we represent the state of a quantum program as a pair $\langle S, \rho \rangle$ where S is the program that still needs to be executed and ρ is the current state of the variables. Logically as we step through a program we execute instructions, and, for each instruction executed, the remaining program changes (shrinks) and the internal variables are modified. We can represent the initial program S prior to execution as $\left\langle S, I_{\text{Var}(S)} \right\rangle$. We can view a program as completed/terminated when it reaches a state of the form $\langle E, \rho \rangle$.

To formalize the notion of *stepping through a program*, we present transition rules from [3] in Figure 1. We observe that these transitions are non-deterministic; a single state could transition to more than one output state. This is intuitive justified by considering a while loop. On a

given iteration of a while loop, it could either terminate or continue to another iteration. A set of transition rules must account for both of these scenarios.

$$\text{Skip:} \quad \frac{}{\langle \mathbf{skip}, \rho \rangle \to \langle E, \rho \rangle}$$

$$\text{Initialization:} \quad \frac{}{\langle q := 0, \rho \rangle \to \langle E, \rho_0^q \rangle}$$

$$\text{Unitary Transformation :} \quad \frac{}{\langle \bar{q} := U\bar{q}, \rho \rangle \to \langle E, U\rho U^\dagger \rangle}$$

$$\text{Composition:} \quad \frac{\langle S_1, \rho \rangle \to \langle S_1', \rho' \rangle}{\langle S_1; S_2, \rho \rangle \to \langle S_1'; S_2, \rho' \rangle}$$

$$\text{Measurement:} \quad \frac{}{\left\langle \mathbf{measure}\ M[\bar{q}] : \bar{S}, \rho \right\rangle \to \left\langle S_m, M_m \rho M_m^\dagger \right\rangle}$$

$$\text{While Loop Termination:} \quad \frac{}{\langle \mathbf{while}\ M[\bar{q}] = 1\ \mathbf{do}\ S, \rho \rangle \to \left\langle E, M_0 \rho M_0^\dagger \right\rangle}$$

$$\text{While Loop Iteration:} \quad \frac{}{\langle \mathbf{while}\ M[\bar{q}] = 1\ \mathbf{do}\ S, \rho \rangle \to \left\langle \mathbf{while}\ M[\bar{q}] = 1, M_1 \rho M_1^\dagger \right\rangle}$$

Figure 1: The transition rules for the while programming language.

We observe that while $\langle \cdot, \cdot \rangle \to \langle \cdot, \cdot \rangle$ is defined by Figure 1, it intuitively represents a transition that occurs in a single step. If there exists an N-sequence of program states $\{\langle S_i, \rho_i \rangle\}_{i=1}^N$ such that $\forall i \in \{1, \ldots, N-1\}$, $\langle S_i, \rho_i \rangle \to \langle S_{i+1, i+1} \rangle$ we say that $\langle S_1, \rho_1 \rangle \to_n \langle S_n, \rho_n \rangle$. If there exists an n such that $\langle S, \rho \rangle \to_n \langle S', \rho' \rangle$ we say $\langle S, \rho \rangle \to_* \langle S', \rho' \rangle$.

3.4 Semantic Function

The semantic function described by [5] formalizes the notion of final states transitioned to by the program S. It is a function that maps from a density operator to the sum of all the density operators that S could transition to in a final state. If more than one unique computational path leads from $\langle S, \rho \rangle$ to $\langle E, \rho' \rangle$, then ρ' will be summed as many times as there are such computational paths. We denote the semantic function of the program S as

$$[|S|] : \mathcal{D}^-(\mathcal{H}_{\mathrm{Var}(S)}) \to \mathcal{D}^-(\mathcal{H}_{\mathrm{Var}(S)}).$$

4 Predicates

Hoare Triplets are the core of Floyd-Hoare logic and they are expressed in the form {P}S{Q}, where P and Q are predicates referred to as preconditions and postconditions respectively, and S is a program. The idea is that if the variables meet the preconditions then after being

transformed by the program they must meet the post condition (assuming the program terminates). Classically, we simply check that the variables satisfy the predicate model. However, this description does not work for non-deterministic computation because variables are in a state with a certain probability so we cannot deterministically say whether or not they are in a certain state. D'Hondt and Panangaden [6] suggest that for probabilistic programming we can consider probability distributions as our states and measurable functions as the predicates. The program modifies the probability distributions and then we check that the expectation values of the measurable functions over the resulting probability distribution for satisfaction. We cannot, however, use this model for quantum computation as they are not consistent with each other. That is because measurable functions do not have to be linear and thus do not necessarily correspond to physical observables.

While we cannot directly use the interpretation of Hoare Triplets from probabilistic computation to quantum computation, it serves as an intuitive foundation to construct one. In essence, a model that reasons about expectation values of physical observables is needed. Since Hermition matrices have real eigenvalues, then their eigenvectors correspond to physical observables and thus we can exploit these matrices to construct the model. D'Hondt and Panangaden do that by making density matrices correspond to the states and Hermitian matrices correspond to the predicates. Thus, if we have a density matrix ρ and a Hermitian matrix H, the trace $tr(H\rho)$ corresponds to the expectation value of physical observables. Hence, we will use the trace to formulate a satisfaction criterion for quantum computation.

Another property that is useful for the correctness model is a complete partial ordering, so we need to add another restriction. The restriction is that the Hermitian matrix representing the predicates must be a positive matrix with its eigenvalues upper-bounded by 1. This follows from the following proposition:

Proposition 1. *For any density matrix ρ and Hermitian operator H we have $0 \leq tr(H\rho) \leq 1$ if and only if H is positive and its eigenvalues are bounded by 1.*

A simple proof of the proposition is provided in [6].

5 Correctness

With a clear definition of states (density matrices) and predicates (Hermitian positive Hermitian operators with eigenvalues bounded by 1), we can begin to define a notion of correctness for quantum programs. As of this point, we assumed that our program terminates. However, this might not be the case. Hence, two models of correctness need to be considered: one model where the program terminates, called total correctness, and another where the program may not terminate, called partial correctness.

5.1 Total Correctness

Consider the post-condition Q, program S, and density function ρ. Intuitively, we want a precondition P to be a predicate such that the expectation value of our post-condition in the

state $[[S]]\rho$ is no less than the expectation value of the precondition in the state ρ. Thus, we get the following definition for total correctness:

Definition 1. *The Hoare Triplet $\{P\}S\{Q\}$ is said to be totally correct, written as:*

$$\models_{tot} \{P\}S\{Q\}$$

if:

$$tr(P\rho) \leq tr(Q[[S]]\rho)$$

As mentioned earlier, the trace of the product of a predicate with a density matrix is the expectation of the predicate. Thus, this definition is consistent with the intuition of correctness provided above.

5.2 Partial Correctness

The intuition for a notion of partial correctness is similar to that of total correctness, but we need to account for the possibility of a program not terminating. So to provide a definition, we must first find the probability of a program diverging on an input ρ. The definition we will provide is specific to a while-program as defined by Ying. In [3], the following proposition is presented:

Proposition 2. *For any quantum program S, it holds that:*

$$tr([[S]]\rho) \leq tr(\rho)$$

While the proposition claims that this is true for any quantum program, the prove inducts on the structure of while-programs. As Ying argues, the inequality comes strictly from the while loops, thus $tr(\rho) - tr([[S]]\rho)$ represents the probability that S diverges on input ρ. Thus, this gives a way of defining partial correctness. However, since the reasoning depends on the structure of while-programs, we restrict the following definition provided in [3] to while-programs S:

Definition 2. *The Hoare Triplet $\{P\}S\{Q\}$ is said to be partially correct, written as:*

$$\models_{par} \{P\}S\{Q\}$$

if:

$$tr(P\rho) \leq tr(Q[[S]]\rho) + (tr(\rho) - tr([[S]]\rho))$$

6 Quantum Weakest Precondition

In classical Hoare-logic, there are preconditions that are referred to as weakest preconditions. The weakest precondition of a program S with postcondition Q is defined as the precondition with the weakest constraint, such that the Hoare triplet $\{P\}S\{Q\}$ holds. The concept of weakest preconditions is important in Hoare-logic as it provides a notion of the very least amount of constraints needed from a precondition, which is crucial for proving this logic model to be complete.

Similarly, there exists a notion of quantum weakest preconditions that helps in building inference rules that provide a complete proof structure. The idea of quantum weakest preconditions was first purposed by D'hondt and Panangaden [6]. The formal definition of quantum weakest predicates is as follows:

Definition 3. *A precondition P is said to be the weakest precondition for a postcondition Q with respect to a program S if:*

$$\models_{tot} \{P\}S\{Q\}$$

$$\models_{tot} \{R\}S\{Q\} \implies R \sqsubseteq P$$

Where $\sqsubseteq$ is the *Löwner* ordering. This ordering is helpful due to the following proposition proved in [6]:

Proposition 3. *$R \sqsubseteq P$ if and only if for all density operators ρ, $tr(R\rho) \leq tr(P\rho)$*

Note that since we chose that the eigenvalues of the predicates to be bounded below by 0 and above by 1, this gives a complete partial ordering.

The weakest precondition for partial correctness is defined analogously.

7 Proof Systems

Using the predicate formulation provided in [6] and the definition of total and partial correctness, Ying is able to provide a proof system for quantum while-programs in [3]. He does so by providing the inference rules in Figure 2 and proving that this system is complete and sound for partial correctness of quantum while programs. The same proof system is used for total correctness with the addition of a rule to account for terminating while loops.

To give a little of intuition behind the prove of completeness, note that when looking at the inference rules in Figure 2, the preconditions provided for a postcondition with respect to a program is the weakest precondition. This allows to replace the weakest precondition with one with stronger constraints using the rule order of needed. Thus, you can begin with the postcondition of a program, apply the inference rules until you reach the beginning of the program, then check if the precondition yielded has weaker constraints than the precondition you want to check. An example of this is provided in the following section.

(Axiom Skip) $\{P\}\mathbf{Skip}\{P\}$

(Axiom Initialization) If type(q) = **Boolean**, then

$$\{|0\rangle_q \langle 0| P |0\rangle_q \langle 0| + |1\rangle_q \langle 0| P |0\rangle_q \langle 1|\}q := 0\{P\}$$

If type(q) = **integer**, then

$$\{\sum_{n=-\inf}^{inf} n_q \langle 0| P0_q \langle n|\}q := 0\{P\}$$

(Axiom Unitary Transformation) $\{U^\dagger PU\}\bar{q} := U\bar{q}\{P\}$

(Rule Sequential Composition) $$\frac{\{P\}S_1\{Q\} \qquad \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}}$$

(Rule Measurement) $$\frac{\{P_m\}S_m\{Q\} \text{ for all m}}{\{\sum_m M_m^\dagger P_m M_m\} \textbf{ measure } M[\bar{q}] : \bar{S}\{Q\}}$$

(Rule Loop Partial) $$\frac{\{Q\}S\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}}{\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}\textbf{while } M[\bar{q}] = 1\textbf{do } S\{P\}}$$

(Rule Order) $$\frac{P \sqsubseteq P' \qquad \{P'\}S\{Q'\} \qquad Q' \sqsubseteq Q}{\{P\}S\{Q\}}$$

Figure 2: Proof System of Partial Correctness

8 Example

We will demonstrate how the described proof system can be used to verify the correctness of a particular program. In this case, we present Ying's analysis of a program which implements the teleportation protocol [1]. A proof of correctness is traced out in Figure 3. Observe that $|\phi\rangle = Z|\varphi\rangle$, $|\theta\rangle = X|\varphi\rangle$, and $|\psi\rangle = XZ|\varphi\rangle$.

We begin by identifying the precondition and postcondition we would like to prove. In our case, we select a precondition of the first qubit being in the designate state $|\varphi\rangle\langle\varphi|$ and the second and third qubit being in the bell state. We select a post condition of $I \otimes I \otimes |\varphi\rangle\langle\varphi|$. This indicates we desire the final qubit to have the value that the first qubit held originally (hence *teleporting* it). We select the identity for the first two qubits because it is the least restrictive predicate. This corresponds to placing no restriction whatsoever on the first two qubits.

After deciding on a post-condition, we can work backwards through the program, recording the weakes precondition at every step. At every step we use the proof system from Figure 2 to

find the weakest precondition for each command.

By the *Axiom Skip* rule we determine the weakest precondition of line 10 to be the same as the postcondition. *Axiom Unitary Transformation* gives the precondition for line 8. By *Rule Measurement*, we transform the weakest preconditions of lines 8 and 10 into the weakest precondition listed for line 7. We can repeat the same process to get the quantum weakest preconditions for lines 3, 4, and 6. Finally, we use the the *Axiom Unitary Transformation* rule two more times to get the weakest preconditions for lines 2 and 1, in that order. We observe that the precondition listed for line 1 is equivalent to $\{|\varphi\rangle_p\langle\varphi|\otimes|\beta_{00}\rangle_{q,r}\langle\beta_{00}|\}$. We therefore conclude that whenever the 2 and 3 qubits are in the bell state and the teleportation protocol is performed, the final qubit will take on the value that the first qubit had orignally. This therefore completes the proof of correctness for the teleportation protocol.

Precondition: $\{|\varphi\rangle_p\langle\varphi|\otimes|\beta_{00}\rangle_{q,r}\langle\beta_{00}|\}$

	Command	**Weakest Precondition**
1	$p, q := \texttt{CNOT}(p, q)$	$\{\|\beta_{00}\rangle_{p,q}\langle\beta_{00}\|\otimes\|\varphi\rangle_r\langle\varphi\|+\|\beta_{10}\rangle_{p,q}\langle\beta_{10}\|\otimes\|\phi\rangle_r\langle\phi\|+$ $\|\beta_{01}\rangle_{p,q}\langle\beta_{01}\|\otimes\|\theta\rangle_r\langle\theta\|+\|\beta_{11}\rangle_{p,q}\langle\beta_{11}\|\otimes\|\psi\rangle_r\langle\psi\|\}$
2	$p := \texttt{H}(p)$	$\{\|+\rangle_p\langle+\|\otimes I_q\otimes\|\theta\rangle_r\langle\theta\|+\|-\rangle_p\langle-\|\otimes I_q\otimes\|\psi\rangle_r\langle\psi\|+$ $\|+\rangle_p\langle+\|\otimes I_q\otimes\|\varphi\rangle_r\langle\varphi\|-\|1\rangle_p\langle-\|\otimes I_q\otimes\|\phi\rangle_r\langle\phi\|\}$
3	**if** $\texttt{M}(q) = 1$:	$\{\|0\rangle_p\langle0\|\otimes I_q\otimes\|\theta\rangle_r\langle\theta\|+\|1\rangle_p\langle1\|\otimes I_q\otimes\|\psi\rangle_r\langle\psi\|+$ $\|0\rangle_p\langle0\|\otimes I_q\otimes\|\varphi\rangle_r\langle\varphi\|+\|1\rangle_p\langle1\|\otimes I_q\otimes\|\phi\rangle_r\langle\phi\|\}$
4	$r := \texttt{X}(r)$	$\{\|0\rangle_p\langle0\|\otimes I_q\otimes\|\theta\rangle_r\langle\theta\|+\|1\rangle_p\langle1\|\otimes I_q\otimes\|\psi\rangle_r\langle\psi\|\}$
5	**else** :	
6	*skip*	$\{\|0\rangle_p\langle0\|\otimes I_q\otimes\|\varphi\rangle_r\langle\varphi\|+\|1\rangle_p\langle1\|\otimes I_q\otimes\|\phi\rangle_r\langle\phi\|\}$
7	**if** $\texttt{M}(p) = 1$:	$\{\|0\rangle_p\langle0\|\otimes I_q\otimes\|\varphi\rangle_r\langle\varphi\|+\|1\rangle_p\langle1\|\otimes I_q\otimes\|\phi\rangle_r\langle\phi\|\}$
8	$r := \texttt{Z}(r)$	$\{I_p\otimes I_q\otimes\|\phi\rangle_r\langle\phi\|\}$
9	**else** :	
10	*skip*	$\{I_p\otimes I_q\otimes\|\varphi\rangle_r\langle\varphi\|\}$

Postcondition: $\{I_p\otimes I_q\otimes|\varphi\rangle_r\langle\varphi|\}$

where $|\phi\rangle = Z|\varphi\rangle$, $|\theta\rangle = X|\varphi\rangle$, $|\psi\rangle = XZ|\varphi\rangle$.

Figure 3: Quantum Weakest Precondition of the Teleportation Protocol [1]

9 Classical Parallel Computation

We now wish to consider Hoare logic and formal verification systems for classical parallel computation [7]. From a program language perspective, the only addition needed to make a programming language parallel would be the addition of a command capable of running two programs concurrently. Here we use the cobegin [...] coend command to run multiple programs

in parallel. For example, if we have three programs (S_1, S_2, and S_3), the syntax for starting all three concurrently would be

$$\mathsf{cobegin}\ [S_1||S_2||S_3]\ \mathsf{coend}.$$

A generally desirable property of concurrent programs is that execution order should not affect the final result. Consider the following two programs.

$$S_1 := x = 0$$

$$S_2 := x = x + 1$$

Here the order of execution will affect the final result. If we execute $S_1; S_2$, we will be left with a postcondition of $x = 1$ for any precondition. If instead we ran $S_2; S_1$, we would have a postcondition of $x = 0$. We refer to this phenomenon as *interference*. In general, it will be advantageous for us to ensure that interference does not occur.

The interference free rule can be used to ensure that two programs do not interfere with each other. Formally we say that S_1 does not interfere with S_2 if

$$\{\mathsf{pre}(S_1) \wedge \mathsf{pre}(S_2)\}S_1\{\mathsf{post}(S_1) \wedge \mathsf{pre}(S_2)\} \quad \texttt{AND} \quad \{\mathsf{pre}(S_1) \wedge \mathsf{post}(S_2)\}S_1\{\mathsf{post}(S_1) \wedge \mathsf{post}(S_2)\}.$$

We say that S_1 and S_2 are *interference-free* if S_1 does not interfere with S_2 and S_2 does not interfere with S_1. We say the a set of programs $\mathbf{S} = \{S_i\}_{i=1}^n$ is interference-free if every pair of programs in the set are interference-free.

If a set of programs is interference free, we gain the following as additional transition rule:

$$\frac{\{P_1\}S_1\{Q_1\}\ \ldots\ \{P_n\}S_n\{Q_n\}\ \wedge (\{S_i\}_{i=1}^n \text{ is interference-free})}{\{P_1 \wedge \ldots \wedge P_n\}\ \mathsf{cobegin}\ [S_1|\ldots|S_n]\ \mathsf{coend}\ \{Q_1 \wedge \ldots \wedge Q_n\}}.$$

10 Towards Parallel Quantum Verification

Our hope is that this survey will serve as the groundwork for the development of a verification system that works for concurrent quantum programs. To this end a major open question associated with the surveyed literature is a method for developing interference-free rules in the quantum program setting. Specifically, in the satisfiability setting of classical predicates we were able to use a logical and to combine two predicates into one. It is currently unclear how exactly to conjugate predicates in the quantum setting.

Furthermore, two concurrent problems are able to interfere with each other even in the absence of shared variables due to the possibility of registers being entangled with each other. Thus, the interference-free rules must be adapted to account for entanglement. However, that would unfortunately not allow us to verify programs using entanglement, which is an important tool in quantum computation. Hence, we think that entanglement might be better addressed by modifying the variable space definition to allow for a more global variable structure.

Finally, while the idea of quantum weakest predicates was helpful in developing the proof system for quantum Hoare-logic, we think that there is room to stray away from the definition of

predicates talked about in [6] and summarized in this paper. That is because while the weakest precondition was helpful for proving completeness, introducing interference-free rules would make the proof system no longer complete regardless. While miss-categorizing more correct programs is problematic, we think that this should be a route of exploration if conjugating the current notion of predicates proves unfeasible.

11 Conclusion

Verification techniques play a large role in computation and thus it is important to develop verification methods for quantum computing, especially with quantum cryptography rising in popularity. Verifying communication protocols is a very crucial capability for quantum cryptography and the existing probabilistic model checking techniques are not good enough [8]. Furthermore, these techniques need to be developed to work for concurrent programming as well due to the inherent parallel structure of quantum programming, as well as the benefit gained by scaling the size of the system via distributed computation.

This survey focused on summarizing the applications of Floyd-Hoare logic in quantum programming. The initial idea comes from D'Hondt and Panangaden's [6] formulation of predicates as positive Hermitian matrices with eigenvalues upper-bounded by 1. Ying was able to use this definition of predicates and the semantic function defined by Sanders and Zuliani [5] to provide a proof system for quantum while-programs. We additionally surveyed the use of interference-free rules in classical parallel computation and addressed some hurdles that make combining these two verification models challenging.

References

[1] Mingsheng Ying. Toward automatic verification of quantum programs. *Formal Aspects of Computing*, pages 1–23, 2016.

[2] Vincent Danos, Ellie D'Hondt, Elham Kashefi, and Prakash Panangaden. Distributed measurement-based quantum computation. *Electronic Notes in Theoretical Computer Science*, 170:73 – 94, 2007. Proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005).

[3] Mingsheng Ying. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems*, 33(6):19, 2011.

[4] R. Chadha, P. Mateus, and A. Sernadas. Reasoning about imperative quantum programs. *Electronic Notes in Theoretical Computer Science*, 158:19 – 39, 2006. Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXII).

[5] Jeff W Sanders and Paolo Zuliani. Quantum programming. In *International Conference on Mathematics of Program Construction*, pages 80–99. Springer, 2000.

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

[7] Krzysztof Apt, Frank S De Boer, and Ernst-Rüdiger Olderog. *Verification of sequential and concurrent programs*. Springer Science & Business Media, 2010.

[8] Simon J Gay, Rajagopal Nagarajan, and Nikolaos Papanikolaou. Qmc: A model checker for quantum systems. In *International Conference on Computer Aided Verification*, pages 543–547. Springer, 2008.