Program Verification for Quantum Algorithms

Pieter Bos, BSc.

Committee:
Prof. Dr. Marieke Huisman
Prof. Dr. Ir. Floris Zwanenburg
Dr. Ir. Marco Gerards

Formal Methods & Tools
University of Twente

June 25, 2021

Abstract

Quantum computers are improving fast, with several companies now presenting quantum computers with dozens of qbits. As quantum computers will likely be scarce for the foreseeable future, the correctness of quantum programs is an important consideration. This research investigates a formal approach to the verification of quantum programs, and evaluates the usability of that approach. Three Hoare-style logics for quantum programs are discussed, and one is used for a case study. A proof of Shor's factoring algorithm is presented in the selected logic. We conclude that while the logic is usable and a proof can be completed, several avenues of improvement are possible.

Contents

Chapter 1

Introduction

Quantum computation is an emerging field that makes big promises: classes of problems can be solved much faster when we base computers on quantum mechanics, rather than classical logic gates. Quantum computers are built from qbits: a special type of memory that stores an exponential amount of information and, crucially, allows us to do computations on that entire state space.

Several companies, such as Google and IBM, boast quantum computers that have dozens of qbits. While this may not sound like much, we have to remember that the information in a quantum computer (measured in bits) expands exponentially with the number of qbits, rather than linearly. We have in fact now reached a point where working quantum computers can no longer be simulated efficiently enough by classical computers to keep up (though this claim is still contested). This is called *quantum supremacy.*

It will still be some time before every household has a quantum computer, if such a time will ever come. For the time being it looks like quantum computers will remain very expensive to make, and require extreme conditions, such as a vacuum and extremely low temperature. It is likely that quantum computers will follow a cloud computing model for some time, where we will be able to interact with quantum computers over the internet.

If quantum computing will remain scarce, it will be important to be confident in the correctness of your quantum program. This is in addition to the fact that it is good to be confident in the correctness of your program in general, especially when we rely upon its correctness for safety. A rigorous approach to ensuring correctness of programs is via formal methods.

The word formal here refers to the fact that we use math to substantiate our claims. We construct an abstract model of the computer, and use that model to prove properties about the program. If we succeed in proving such properties, we can be very confident that our program will respect that property. One such type of model uses Hoare logic, in which we can be very precise about the inputs a program accepts, and the output it will produce. This is the type of logic that this thesis will deal with.

1.1 Objective

This thesis investigates the usability of a Hoare-style logic for quantum programs. The formal research questions are:

Q1. What is a representative example to show the usability of quantum logics?

Q2. How can the example be proven in existing logics?

Q3. How usable are existing logics for the verification of quantum programs?

Q4. Could the logics be adapted to be included in automated proof tools?

The example chosen for Q1 is Shor's factoring algorithm. It is a well-known algorithm that produces a factor of a composite number. The primary motivation to choose this algorithm is that it contains both a non-trivial classical part and quantum part. The algorithm is further motivated and explained in Chapter 6.

The central proof of this work shows that an implementation of Shor's algorithm in a theoretical quantum programming language is correct: it produces a factor of a composite number. The proof for Q2 is carried out in Chapter 8. As for usability in Q3: a fair number of lemmas was needed to complete the proof, as is reflected in Chapter 7. This is also discussed in the findings in Chapter 9.

Finally the end goal is to be able to construct tools that make proving properties about quantum programs easy. For Q4 this is briefly reflected on in the conclusion, and is left for future work.

1.2 Structure

A large part of the thesis is to set up the necessary background to do a formal proof of a quantum algorithm. First background is given on program verification in Chapter 2, and quantum computers in Chapter 3. An embedding of quantum computation in a logic constitutes a quantum program logic, examples of which are given in Chapter 4. We choose one logic to explain more deeply in Chapter 5, to be able to do a proof in it. The case study for the proof is explained in Chapter 6. Finally, we carry out the proof by first giving some lemmas in Chapter 7, and then constructing the proof in Chapter 8.

Chapter 2

Program Verification

This chapter gives background on a technique for program verification. In particular, we show how to design a Hoare-style proof system. We later need this background together with Chapter 3 to reason about logics for quantum programs in Chapter 4.

This thesis deals with showing the correctness of programs. To do this, we have to define some measure of correctness. In particular, we would like our correctness to be mathematically rigorous, so we can be sure that a program is correct.

We use derivatives of Hoare logic [7], which presents an intuitive way to reason about programs. The intuition is that we first require the computer to be in some state (the *precondition*, ϕ), then we execute the program P, and then we promise something about the state of the computer (the *postcondition*, ψ). This is then denoted as a Hoare triple:

$$\{\phi\}\ P\ \{\psi\}$$

How we represent the state varies by logic, but one might imagine it describes the memory of the computer, or the variables used in the program. For example, a specified program that assigns a value to a variable x might look like the following:

$$\{x = 0\}\ \texttt{x := 2}\ \{x = 2\}$$

This then states: for every program state where $x = 0$, after executing `x := 2`, we know that $x = 2$. That is of course too strict to be general: x can also be 1 before the assignment, and it will still be 2 afterwards. We can therefore broaden the precondition to not require anything about x:

$$\{\mathsf{true}\}\ \texttt{x := 2}\ \{x = 2\}$$

To make this more precise, we should specify two things: we need to know when a state satisfies a pre- or postcondition, and we should know how a program affects the state.

$$\models\ :\ (\mathsf{state} \times \phi) \to \mathbb{B} \tag{2.1}$$

$$[\![\cdot]\!](\cdot)\ :\ (\mathsf{program} \times \mathsf{state}) \rightharpoonup \mathsf{state} \tag{2.2}$$

The symbol '$\models$' in (2.1) is used for the relation between states and predicates (pre- and postconditions). We say that a state satisfies ϕ when $\mathsf{state} \models \phi$.

The operator '$[\![\cdot]\!](\cdot)$' in (2.2) is used to enact a program on a state. Given an input state S and a program P, the result of executing P in state S is again a state, and is denoted: $[\![P]\!](S)$. This operator is said to define the *semantics* of programs. It is partial, as defined in (2.2), because some programs never complete.

A Hoare logic is assembled as a collection of *proof rules* and *axioms*. A proof rule for two programs executed directly after each other typically looks like this:

$$\frac{\{\phi\}\ P\ \{\phi'\} \qquad \{\phi'\}\ Q\ \{\psi\}}{\{\phi\}\ P\texttt{;}\ \ Q\ \{\psi\}}\ \textsc{seq}$$

This rule simply states that if $\{\phi\}\ P\ \{\phi'\}$ holds, and $\{\phi'\}\ Q\ \{\psi\}$ holds, then $\{\phi\}\ P\texttt{;}\ Q\ \{\psi\}$ must hold. Whether the logic itself is correct can then be established by checking that we can only prove things that are true with respect to the semantics of the program, and the satisfaction relation for predicates. That is, we can only prove $\{\phi\}\ P\ \{\psi\}$ if $S \models \phi$ implies $[\![P]\!](S) \models \psi$. If this always holds, we say the logic is *sound*. The opposite property also has a name: a logic is said to be *complete* when for every P, S, ϕ, ψ where $S \models \phi$ implies $[\![P]\!](S) \models \psi$, there is a proof leading to $\{\phi\}\ P\ \{\psi\}$.

2.1 WHILE

To give an intuition on how a semantics might be structured, we give an example. It is usual for the *syntax* of programs in a logic to be just powerful enough to express useful programs, but restrained so that the semantics and proof rules do not become unnecessarily cluttered. One such standard language is WHILE. The syntax of WHILE is:

$$\begin{aligned} P ::=\ & \texttt{skip} \\ \mid\ & x\ \texttt{:=}\ E \\ \mid\ & P\texttt{;}\ \ P \\ \mid\ & \texttt{if}\ B\ \texttt{then}\ P\ \texttt{else}\ P \\ \mid\ & \texttt{while}\ B\ \texttt{do}\ P \end{aligned} \tag{2.3}$$

$$E ::= n \in \mathbb{N} \mid x \in X \mid f(E, \ldots, E) \tag{2.4}$$

$$B ::= E \leq E \mid B \wedge B \mid \neg B \tag{2.5}$$

Here x ranges over some arbitrary set of variable symbols X, and f denotes total functions. The state used to describe the semantics of WHILE programs is a function $S : X \to \mathbb{N}$. We can now define the semantics of WHILE:

$$\begin{aligned} [\![\texttt{skip}]\!](S) &= S \\ [\![x\ \texttt{:=}\ E]\!](S) &= S[x \mapsto [\![E]\!](S)] \\ [\![P\texttt{;}\ \ Q]\!](S) &= [\![Q]\!]([\![P]\!](S)) \\ [\![\texttt{if}\ B\ \texttt{then}\ P\ \texttt{else}\ Q]\!](S) &= \begin{cases} [\![P]\!](S) & \text{if } [\![B]\!](S) \\ [\![Q]\!](S) & \text{otherwise} \end{cases} \\ [\![\texttt{while}\ B\ \texttt{do}\ P]\!](S) &= \begin{cases} [\![\texttt{while}\ B\ \texttt{do}\ P]\!]([\![P]\!](S)) & \text{if } [\![B]\!](S) \\ S & \text{otherwise} \end{cases} \end{aligned} \tag{2.6}$$

In a slight abuse of notation $[\![E]\!](S)$ and $[\![B]\!](S)$ are also used to interpret expressions and conditions in a state:

$$\begin{aligned}
[\![n \in \mathbb{N}]\!](S) &= n \\
[\![x \in X]\!](S) &= S(x) \\
[\![f(E_1, \ldots, E_n)]\!](S) &= f([\![E_1]\!](S), \ldots, [\![E_n]\!](S)) \\
[\![E_1 \leq E_2]\!](S) &= [\![E_1]\!](S) \leq [\![E_2]\!](S) \\
[\![B_1 \land B_2]\!](S) &= [\![B_1]\!](S) \land [\![B_2]\!](S) \\
[\![\neg B]\!](S) &= \neg[\![B]\!](S)
\end{aligned} \tag{2.7}$$

2.2 Axiomatic Semantics

We can now start to design a set of proof rules and axioms that correspond to our semantics of WHILE. For example, we can declare an axiom about `skip`:

$$\frac{}{\{\phi\}\ \texttt{skip}\ \{\phi\}}\ \textsc{skip} \tag{2.8}$$

This axiom is sound: clearly $S \models \phi$ implies $[\![\texttt{skip}]\!](S) = S \models \phi$. We can also inspect the completeness of this single-rule proof system, supposing for the moment that the only possible program is `skip`. If we can find ϕ and ψ such that $S \models \phi$ implies $[\![\texttt{skip}]\!](S) \models \psi$, and we cannot prove this with our proof system (i.e. the one axiom), the proof system is not complete. This is rather easy to show: a counterexample is $\{\mathsf{false}\}$ `skip` $\{\mathsf{true}\}$. This statement is true, as $S \not\models \mathsf{false}$ for any S, and so the constraint $S \models \mathsf{false} \Rightarrow [\![\texttt{skip}]\!](S) \models \mathsf{true}$ is satisfied. The triple $\{\mathsf{false}\}$ `skip` $\{\mathsf{true}\}$ does not follow from the axiom, as false is not the same predicate as true.

To make this logic complete (for the program `skip`), at a minimum we have to add a proof rule that says something about Hoare triples with an inequal pre- and postcondition. Specifically, the rule we still need is one that can strengthen preconditions (i.e. it excludes more states), and a rule that can weaken post-conditions (i.e. it allows for more states):

$$\frac{\{\phi'\}\ P\ \{\psi\} \qquad \phi \to \phi'}{\{\phi\}\ P\ \{\psi\}}\ \textsc{pre-logic} \tag{2.9}$$

$$\frac{\{\phi\}\ P\ \{\psi'\} \qquad \psi' \to \psi}{\{\phi\}\ P\ \{\psi\}}\ \textsc{post-logic} \tag{2.10}$$

We can again show these rules are sound. For PRE-LOGIC we have to show that given an $S \models \phi$, $[\![P]\!](S) \models \psi$. We can see that $S \models \phi'$, as otherwise $S \not\models \phi \to \phi'$. From $\{\phi'\}\ P\ \{\psi\}$ we can then conclude that $[\![P]\!](S) \models \psi$. Rule POST-LOGIC has a similar proof.

Our counterexample now no longer works: we can use $\mathsf{false} \to \mathsf{true}$ to prove $\{\mathsf{false}\}$ P $\{\mathsf{true}\}$:

$$\frac{\dfrac{}{\{\mathsf{true}\}\ \texttt{skip}\ \{\mathsf{true}\}}\ \textsc{skip} \qquad \mathsf{true} \to \mathsf{false}}{\{\mathsf{false}\}\ \texttt{skip}\ \{\mathsf{true}\}}\ \textsc{pre-logic} \tag{2.11}$$

In fact, we can now prove the logic is complete for the program `skip`. Given any $\{\phi\}$ `skip` $\{\psi\}$, we have $S \models \phi \Rightarrow [\![P]\!](S) = S \models \psi$. Clearly then S satisfies $\phi \to \psi$, and so we can build a proof tree for any $\{\phi\}$ `skip` $\{\psi\}$:

$$\dfrac{\dfrac{}{\{\psi\}\ \texttt{skip}\ \{\psi\}}\ \textsc{skip} \qquad \phi \to \psi}{\{\phi\}\ \texttt{skip}\ \{\psi\}}\ \textsc{pre-logic} \tag{2.12}$$

While completeness proofs are interesting, for the remaining syntactical constructs the proofs get quite involved, so we only prove soundness. The next piece of syntax is assign:

$$\dfrac{}{\{\phi[e/x]\}\ x\ \texttt{:=}\ e\ \{\phi\}}\ \textsc{assign} \tag{2.13}$$

By $\phi[e/x]$ we mean ϕ where every occurrence of x is replaced by e. Intuitively this says that anything that is true for e in the precondition, is true for x in the postcondition. For example, we can compute the precondition for the form $\{\dots\}$ `x := 5` $\{x = 5\}$. We have to replace every occurence of x in $x = 5$ with 5, yielding $\{5 = 5\}\ x := 5\ \{x = 5\}$.

This rule is also sound, since the rule matches the definition of the semantics of assignment: $S \models \phi[e/x]$ iff $S[x \mapsto [\![e]\!](S)] \models \phi$. That is: it is equivalent to replace x with e in the formula, and to assign e to x in the state. A precise proof of this requires a proof by *structural induction* on the formula ϕ. The intent is however not to provide a precise axiomatization of WHILE, but rather to sketch how a logic is structured in general, and how it might be used, and so we skip further detailed proofs.

Next is sequential composition. For this rule we have to provide a formula that holds inbetween two statements that are executed after each other. When the postcondition of the first statement and the precondition of the second statement match, we can connect them:

$$\dfrac{\{\phi\}\ P\ \{\phi'\} \qquad \{\phi'\}\ Q\ \{\psi\}}{\{\phi\}\ P\texttt{;}\ Q\ \{\psi\}}\ \textsc{seq} \tag{2.14}$$

To show soundness, we can use the two hoare triples that we know hold:

$$\forall S : S \models \phi \Rightarrow [\![P]\!](S) \models \phi', \quad \forall S : S \models \phi' \Rightarrow [\![Q]\!](S) \models \psi$$

We can therefore conclude that for all S: $S \models \phi$ implies $[\![Q]\!]([\![P]\!](S)) \models \psi$. This matches the semantics of sequential composition, and so $S \models \phi$ implies $[\![P\texttt{;}\ Q]\!] \models \psi$.

Lastly we have the two branching instructions, the first of which is `if`. In the branch when the condition is true, we can copy the precondition of the `if` statement, and add that the condition is true. Conversely in the branch where the condition is false we may add that the condition is false. To be able to join the branches again, their postconditions should match:

$$\dfrac{\{\phi \wedge c\}\ T\ \{\psi\} \qquad \{\phi \wedge \neg c\}\ F\ \{\psi\}}{\{\phi\}\ \texttt{if}\ c\ \texttt{then}\ T\ \texttt{else}\ F\ \{\psi\}}\ \textsc{if} \tag{2.15}$$

Here the core insight is that we can move the condition out of the formula in the branches: $S \models \phi \wedge c$ iff $[\![c]\!](S) \wedge S \models \phi$, and similarly $S \models \phi \wedge \neg c$ iff $\neg[\![c]\!](S) \wedge S \models$

ϕ. Then we can simplify unfold the semantics of `if` into its two branches:

$$\begin{aligned} & S \models \phi \Rightarrow \left([\![\texttt{if } c \texttt{ then } T \texttt{ else } F]\!](S) = \begin{cases} [\![T]\!](S) & \text{if } [\![c]\!](S) \\ [\![F]\!](S) & \text{otherwise} \end{cases} \right) \models \psi \\ & \text{iff } (S \models \phi \wedge [\![c]\!](S) \Rightarrow [\![T]\!](S) \models \psi) \wedge (S \models \phi \wedge \neg [\![c]\!](S) \Rightarrow [\![F]\!](S) \models \psi) \\ & \text{iff } (S \models \phi \wedge c \Rightarrow [\![T]\!](S) \models \psi) \wedge (S \models \phi \wedge \neg c \Rightarrow [\![F]\!](S) \models \psi \end{aligned}$$

The other branching statement is `while`. For loops we use the concept of a *loop invariant*: a condition which must be maintained for every iteration of the loop. For the loop body we then want to prove that if the invariant holds before the body (and the condition is true), it also holds after execution of the body. If this is the case the invariant must also be true after the whole loop (and the condition will be false).

$$\frac{\{\phi \wedge c\}\ P\ \{\phi\}}{\{\phi\}\ \texttt{while}\ c\ \texttt{do}\ P\ \{\phi \wedge \neg c\}}\ \text{WHILE} \tag{2.16}$$

For soundness we want:

$$S \models \phi \Rightarrow \left(\begin{cases} [\![\texttt{while } c \texttt{ do } P]\!]([\![P]\!](S)) & \text{if } [\![c]\!](S) \\ S & \text{otherwise} \end{cases} \right) \models \phi \wedge \neg c \tag{2.17}$$

When $[\![c]\!](S)$ is false, clearly S satisfies $\phi \wedge \neg c$. On the other hand when $[\![c]\!](S)$ is true, we use $\{\phi \wedge c\}\ P\ \{\phi\}$ to obtain $[\![P]\!](S) \models \phi$. If we assume that the loop as a whole terminates in some number of iterations, we can then prove by induction that the loop as a whole will maintain ϕ.

Chapter 3

Quantum Computing

This chapter gives background on the mechanics of quantum computing. We use the conditions set out by DiVincenzo in [6] as a guide to an abstract mathematical model of quantum systems that we can use to do quantum computation. Together with the background on program verification logic from Chapter 2 this chapter is used to reason about quantum program logics in Chapter 4.

Computers are based on the fact that there exist physical phenomena that we can exploit to automate logic. Modern computers are based on semi-conductors, which we use to construct transistors, and in turn logic gates. Computers are only about as complex as the number of logic gates it has, and the number of computations it can do with them per unit of time. Given enough paper, pens, and patience, a human could do the same computation in a constant multiple of time that the computer can do the computation in.

Quantum computers on the other hand are based on a physical process that does not scale linearly in the same way. The goal is to find particles that can act as a *qbit*. Qbits compose into a *quantum system* where the state space grows exponentially with the number of qbits, as well as the atomic operations we can perform on the state space. It is important to note that we cannot perform arbitrary operations on a quantum system, and correspondingly quantum computing does not trivially provide an exponential speed-up for every algorithm.

A more precise characterization of quantum computers is given by DiVincenzo as five criteria in [6]. These criteria are enumerated and explained below to motivate the choices made in the quantum computing language presented in Chapter 5.

3.1 Conditions for Quantum Computing

"A scalable physical system with well characterized [qbits]": a qbit is a quantum system (for example: one particle), that can be in two states. The two states are usually denoted $|0\rangle$ and $|1\rangle$. The crucial properties that quantum computing uses is *superposition* and *entanglement*. Superposition refers to the fact that a qbit is generally not in a *ground state*, in this case either $|0\rangle$ or $|1\rangle$. In actuality, the qbit has some probability of ending up in either state. Entanglement refers to the fact that the probability of ground states of two entangled qbits cannot be considered individually. Two entangled qbits have some probability of ending up as $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. It is important to note that these properties correspond to the physical reality that a quantum system can be in a superposition, rather than that we prefer to model the system probabilistically. If we start with the presumption that in actuality the quantum system must have started in some ground state, we are bound to arrive at a contradiction: an entangled quantum system in a superposition simply stores more information than a ground state, which a quantum computer can demonstrate. Well characterized refers to several physical properties, of which computer scientists may hope they do not have to deal with. For example, if a qbit has a secret third state, the probability of ending up in it should be low. Scalable refers to the number of qbits we can successfully entangle.

"The ability to initialize the state of the [qbits] to a simple fiducial state, such as $|000\ldots\rangle$": Though normally the quantum system is in a superposition, we want to be able to control the state that it starts in. Thus, we need the ability to initialize the quantum system into one known ground state.

"A [qbit]-specific measurement capability": A good measurement is usually thought of as to interfere with the system it measures as little as possible. For quantum systems the opposite is true: measurement of a qbit makes it so that the qbit is thereafter in a ground state, corresponding to the measurement outcome. Measurement is thus also understood to be an action on the system. For quantum computation to work, it is important that we are able to measure only one or a set of qbits, rather than the whole system.

"A 'universal' set of quantum gates": Quantum gates are the computational operation on quantum systems, drawing on the analogy of a logic gate. Quantum algorithms often contain operations on a large part of the system, e.g. a Fourier transform on a set of qbits. These are often not implemented directly, but composed from a set of universal quantum gates. The term universal means that the set is sufficient to construct any allowed transformation on the quantum system.

"Long relevant decoherence times, much longer than the gate operation time": Decoherence is a specific physical process, where the state of a quantum system decays into a different state. For a computer it is quite undesirable that its state spontaneously evolves. The exact mechanics of decoherence are not further discussed here.

3.2 An Abstract Quantum Computer

To be able to reason about quantum computers, we have to make some assumptions. First and foremost, we assume that there are no error conditions for the quantum computer: any invalid states are unreachable. Furthermore, there are none of the usual spontaneous effects on quantum systems: no decoherence, no measurement unless we specify it, and operations perform exactly the transformation we prescribe.

These assumptions map well to the assumptions normally made about regular computers: in formal methods, effects like electrical noise are not considered. We choose an abstraction layer and stick with that, and so we do the same here. Now that we have specified how a quantum computer should work, and what assumptions we make, we can construct a mathematical model of it.

3.3 Mathematical Preliminaries

As said before, the true state of a quantum system is probabilistic. For example, in a 5-qbit system the system has 2^5 possible ground states. To capture the state of the quantum system, we have to store the probability of each ground state. Correspondingly a quantum system is modeled as a vector space, with a dimension per ground state. For ground states we use a shorthand notation called a 'ket', part of Dirac notation [5]:

$$|0\rangle \equiv e_0 \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle \equiv e_1 \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.1}$$

Also in common use is the 'bra', which is the transpose of the 'ket':

$$\langle \phi | \equiv |\phi\rangle^T \tag{3.2}$$

The vector space is over $\mathbb{C}$, where a scalar is called a *probability amplitude*. When we say that a probability corresponds to a ground state, what we really mean is that when we *measure* the system, that is the probability we will measure that state. This does not mean that the system was secretly in that state the entire time: it remains in an ensemble of ground states until we measure it. To obtain the probability assigned to a ground state, we project the system to its corresponding dimension, and take the square of its 2-norm. For a one-qbit system, the projection operators are:

$$E_0 \equiv \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad E_1 \equiv \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.3}$$

For a system A we can then obtain the probability of measurement outcome 0 as $|E_0 A|^2$, and the probability of measurement outcome 1 as $|E_1 A|^2$. A measurement is also an action on the system, namely the same projection to a ground state. In particular, when we measure a 0, the system will then be in state $E_0 A / |E_0 A|^2$. When we measure a 1, the system will then be in state $E_1 A / |E_1 A|^2$.

Of course we are interested in multi-qbit systems, so we want to compose them. We might enumerate all possible ground states manually again, as such:

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Since the dimension of the vector space grows exponentially with the number of qbits, this quickly becomes unwieldy. Luckily, we can use the tensor product to do the work for us instead, defined for two matrices as such:

$$A_{m \times n} \otimes B_{o \times p} = \begin{bmatrix} a_{11} \cdot B & \dots & a_{1n} \cdot B \\ \vdots & \ddots & \vdots \\ a_{m1} \cdot B & \dots & a_{mn} \cdot B \end{bmatrix} \tag{3.4}$$

The B matrix is expanded in place for each element of A, resulting in a matrix in $\mathbb{R}^{(m \cdot o) \times (n \cdot p)}$. We can now define $|00\rangle$ as $|0\rangle \otimes |0\rangle$, $|000\rangle$ as $|0\rangle \otimes |0\rangle \otimes |0\rangle$, etc.

Of course we cannot apply our projection operators in (3.3) to multi-qbit systems, but we can expand them by taking the tensor product with I. In this way, the project operator only projects the qbit we are interested in, while not measuring other qbits. Due to entanglement a qbit that is not measured may still be affected. For example: in the system $(|01\rangle + |10\rangle)/\sqrt{2}$ measuring either qbit affects the other qbit. To project the second qbit in a three-qbit system to $|0\rangle$, the operator would be $I \otimes E_0 \otimes I$. Usually qbits, or collections of qbits, are named and in some implied order, in which case we may write a projection operator as ${}^{a}E_0$, which is understood to measure qbit a and is in some form $I \otimes \ldots \otimes E_0 \otimes \ldots \otimes I$.

For reasons not explained in detail here it is sometimes more natural to represent a quantum system with a *density matrix* instead. Whereas the state ϕ was earlier represented as $|\phi\rangle$, we now represent it as $|\phi\rangle\langle\phi|$. That is: we take the original representation, and take the outer product with itself.

The state of a quantum system should obey the laws of probability. To that end density matrices must obey three properties: it is *positive semi-definite*, *Hermitian*, and has *trace* 1.

- A *Hermitian* matrix is defined to be equal to its conjugate transpose, by convention written as $A^\dagger \equiv \overline{A^T}$.
- The *trace* of a matrix is the sum of its diagonal: $tr(A_{n\times n}) \equiv \sum_{i=1}^{n} A_{ii}$.
- A matrix $A_{n\times n}$ is said to be *positive semi-definite* when for all $x \in \mathbb{C}^n$: $x^\dagger A x \geq 0$.

A closely related concept is that of a *partial* density matrix. In a regular density matrix, the trace being one is a reflection of the fact that the sum over all ground states of the probability of measuring that ground state must be one. In a partial density matrix this restriction is replaced by the restriction that the trace must be less than or equal to one. It is implicitly greater than or equal to zero because of the fact that the matrix is positive semi-definite. This is helpful, since this allows us to perform a probabilistic branch. For example, after measuring a qbit we can consider the case that it is now in ground state $|0\rangle$ and in ground state $|1\rangle$ separately (i.e. without normalization), and later join them by adding them together, which gives us a density matrix again.

Finally, we introduce some extra notation as syntactic sugar:

- $|n \in \mathbb{N}\rangle$ implicitly translates n into binary, with some implied width. For example, for a four-qbit register we have $|5\rangle = |0101\rangle$.
- ${}^{a}E_{n\in\mathbb{N}}$ similarly measures multiple qbits, where n is translated into binary.

Chapter 4

Related Work

This thesis evaluates the usability of an existing quantum logic. In this chapter we explore some works that propose a logic for the verification of quantum programs, in order to select one to do a case study in. The chosen logic is explained more deeply in Chapter 5. All three papers listed below ([9], [13], and [11]) give Hoare-like logics for verification of quantum programs, using different formalisms for quantum programs and predicates in the Hoare logic. Their explanation relies on the background on program verification set out in Chapter 2 and quantum computation in Chatper 3. However, we will give a broad intuition here to what a quantum program might look like and how that corresponds to reality.

Firstly, it is important to note that physical quantum computers tend to be imperative. The quantum system is composed of some number of qbits, to which we can apply operators. It is imperative in the sense that we must enact some change on the system to transition it to another state; the system is stable under no operation. Therefore it makes some sense to choose an imperative programming model for quantum programs.

The approach each of the related works has chosen is to embed quantum operators in a classical imperative language. That is, we choose a language that has variables, conditionals, loops, etc. and add statements that can interact with a quantum system. Thus, a quantum algorithm transforms a given problem into a format where a quantum system can provide a speed-up: initialize the quantum system in some condition, perform transformations on the system and then measure the result. The result is then transformed back to a useful answer. As quantum algorithms are often probabilistic, the process is often repeated. This is either because the answer was known-wrong, or to increase the confidence that the answer is right if this cannot be ascertained easily with classical computing.

For each paper we will discuss the quantum program language used and its informal semantics, as well as the logic used for verification.

4.1 A Logic for Formal Verification of Quantum Programs (2009) [9]

4.1.1 Program Syntax and Semantics

This logic uses an adaptation of WHILE, the language introduced in Section 2.1. Programs in this syntax have a mixed state space of bits and qbits. This means it can contain variables as in regular programming languages, as well as named qbits that span a quantum system we can interact with.

Firstly, a statement is added that can initialize an individual qbit, to either 0 or 1. This is different from the requirement specified earlier in [6], where we required that we can initialize the whole system to a known state. Of course we can still do that within the language by assigning all of the qbits some state, but the reverse is not true: we cannot force a specific qbit into a state while retaining the coherence with other qbits. Furthermore, a statement is added that applies a unitary transformation to a set of qbits.

Secondly, there are two branching statements: `measure` and `if`. `if` is a classical boolean branch, whereas `measure` first performs a measurement, and then performs a classical branch based on the measured value. `if` just reads a value from the state space, `measure` also acts on it by forcing a qbit into a ground state.

Finally, the set of declared variables and their type (bit or qbit) is carried through the program as well. There is also a statement that drops a variable so it is no longer available for use.

In the denotational semantics, the state is carried through the program as a density operator, with the probability encoded in the density operator.

4.1.2 Hoare Logic

Predicates in Hoare logic reason about the state of a program, which in our case is a partial density operator. This logic uses first order logic, with these extensions:

- $pr(x)$, meaning the probability that x holds after measurement of all qbits.
- $\Phi_1 \oplus \Phi_2$ is satisfied by a partial density operator ρ, when it can be split as $\rho = \rho_1 + \rho_2$, such that ρ_1 satisfies Φ_1 and ρ_2 satisfies Φ_2. This operator is borrowed from den Hartog [4].
- $t\Phi$ is satisfied by ρ when there is some ρ' for which $t \cdot \rho' = \rho$ and ρ' satisfies Φ. In other words, this operator scales the partial density operator.
- $M\Phi$ is the unitary transformation of Φ with M.

An example that highlights some of the operators is:

$$\frac{\{E_0\phi\}\ P_0\ \{\psi_0\} \qquad \{E_1\phi\}\ P_1\ \{\psi_1\}}{\{\phi\}\ \texttt{measure}\ q\ \texttt{then}\ P_1\ \texttt{else}\ P_0\ \{\psi_0 \oplus \psi_1\}}\ \textsc{Measure}$$

Here $E_0\phi$ and $E_1\phi$ are not normalized, such that $E_0\phi \oplus E_1\phi = \phi$, as E_0 and E_1 represent all measurement outcomes. We then combine the conclusions ψ_0 and ψ_1 with $\oplus$.

4.2 Floyd–Hoare Logic for Quantum Programs (2011) [13]

4.2.1 Program Syntax and Semantics

This logic also uses a version of WHILE. These are the adaptations made to make it work in the quantum case:

- The state space is replaced by a (possibly infinite) set of qbits. q below is used to denote a quantum register, which is a set of qbits.
- Assignment is replaced by two statements: `q := 0` (initialization) and `q := Uq` (unitary transformation).
- `while` and `if` are replaced by similar statements, where the conditional is replaced by a measurement on a quantum register, denoted $M[q]$. The `if` statement is associated with a number of branches equal to the number of measurement outcomes, whereas the `while` condition must have only two measurement outcomes.

The operational semantics makes no presumption on the initial values of the state space, so the state is carried through the program as a partial density operator. Initialization and unitary transformation are defined by applying the statement to the partial density operator. Sequential composition is defined in the conventional way. The if-like operator is called `measure` and is defined as follows:

$$\frac{}{\langle \textbf{measure}\, M[q] : S, \rho\rangle \rightarrow \langle S_m, M_m \rho M_m^\dagger\rangle}\ \textsc{Measurement}$$

One might expect a probabilistic transition here with probability $p(m) = tr(M_m \rho M_m^\dagger)$ and subsequent state $M_m \rho M_m^\dagger / p(m)$. The author has chosen to instead fold the probability into the partial density operator, modeling choice as non-determinism instead. `while` is defined similarly:

$$\frac{}{\langle \textbf{while}\, M[q] = 1\ \textbf{do}\, S, \rho\rangle \rightarrow \langle E, M_0 \rho M_0^\dagger\rangle}\ \textsc{Loop 0}$$

$$\frac{}{\langle \textbf{while}\, M[q] = 1\ \textbf{do}\, S, \rho\rangle \rightarrow \langle S;\ \textbf{while}\, M[q] = 1\ \textbf{do}\, S, M_1 \rho M_1^\dagger\rangle}\ \textsc{Loop 1}$$

Note the non-determinism again, where the while is duplicated again in the regular way when the condition is measured to be true.

4.2.2 Hoare Logic

Predicates say something about the state of a program, which in our case is a partial density operator. Here, predicates are defined as operators, subject to some conditions that will not be further discussed here. The idea is to constrain the partial density operator such that we measure only the probability over desired situations, e.g. certain measurement outcomes.

$$\begin{aligned}
&\models_{\text{total}} \{P\}\ S\ \{Q\} && \text{iff}\ tr(P\rho) \leq tr(Q[\![S]\!](\rho))\\
&\models_{\text{partial}} \{P\}\ S\ \{Q\} && \text{iff}\ tr(P\rho) \leq tr(Q[\![S]\!](\rho)) + (tr(\rho) - tr([\![S]\!](\rho)))
\end{aligned}$$

The trace operator of a partial density operator represents the total probability encoded in the partial density operator, so $tr(P\rho)$ can be understood to be the probability of P being 'true' within the density operator. Consequentially, the properties express that Q is at least as likely as P to be true after the program has executed. In the partial case we simply add the probability of the program diverging.

4.3 Quantum Relational Hoare Logic (2018) [11]

This logic does not reason about single quantum programs, but instead about the equivalence of two quantum programs.

4.3.1 Program Syntax and Semantics

The program syntax and semantics are very similar to the previous two definition adapting WHILE, except that measurement is done as an action separate from `while` or `if` constructs, and assigned to a classic variable. The classic variable may then be used in a condition.

The operational semantics is defined over a partial density operator again. The program equivalence part is only dealt with after the operational semantics, but the general idea is that both related programs can advance independently. As with the other logics, the probabilistic part of the semantics is folded into the partial density operator. The language also contains a classically probabilistic sampling operator ('random'), which is again folded in the partial density operator.

4.3.2 Hoare Logic

Predicates are represented by a subspace of the of the state space. A predicate A is then satisfied by a partial density operator when $\mathbf{supp}\,\rho \subseteq A$. This means that the predicate essentially describes a set of pure states that we allow: if we would measure the system, the chance that the state of the quantum system is not in A is zero. Of the three logics described, this is the least general predicate.

The Hoare triples in the logic borrow from classical relational Hoare logic [2], written here as $\{A\}\ c \sim d\ \{B\}$. This intuitively claims that if A holds before execution of either c or d, both programs behave equivalently by ensuring B.

4.4 Logic for the Case Study

The logic selected for the case study is that introduced in [9]. Initially we proposed to do a case study in all three logics described in this chapter (i.e.

formulate a proof for an equivalent program in each proof system), but due to time constraints one proof was completed.

The logic is however a logical starting point. In particular we want to demonstrate the correctness of a quantum program, and so a relational Hoare logic such as the one in [11] is not an obvious fit. Between [9] and [13] the formulae admitted in preconditions and postconditions made the difference. In the former a formula language is introduced to reason about the state, whereas the latter resorts to an operator over the quantum system. The choice was thus made to do the case study in [9], which is explained in depth in Chapter 5.

Chapter 5

Proof System

This chapter contains an overview of the syntax of the quantum programming language [9], as well as a listing of relevant proof rules. Moreover, the syntax for formulae and their meaning is also discussed formally, as we need some additional lemmas in the proof of Shor's algorithm.

5.1 Syntax

The syntax is defined as an extension of WHILE:

$$\begin{aligned} P \equiv\ & \texttt{skip} \mid P\texttt{;}\ P \\ & \mid \texttt{bit}\ x \mid \texttt{qbit}\ x \mid \texttt{discard}\ x \\ & \mid x\ \texttt{:= 0} \mid x\ \texttt{:= 1} \mid x, \ldots, x\ \texttt{*=}\ U \\ & \mid \texttt{if}\ x\ \texttt{then}\ P\ \texttt{else}\ P \\ & \mid \texttt{while}\ x\ \texttt{do}\ P \\ & \mid \texttt{measure}\ x\ \texttt{then}\ P\ \texttt{else}\ P \end{aligned}$$

`bit` x and `qbit` x respectively declare a bit and qbit. `discard` x removes a bit or qbit from the typing context, which we do not consider. `measure` is the quantum equivalent of `if`: it measures a qbit and branches on the result. Finally, $x, \ldots, x$ `*= U` performs a unitary transformation on qbits $x, \ldots, x$.

We also use some of the defined syntax sugar:

$\texttt{bit}\ x\texttt{[}N\texttt{]} \equiv \texttt{bit}\ x\texttt{[0]; }\ldots\texttt{; bit}\ x\texttt{[}N-1\texttt{]}$

$\texttt{qbit}\ x\texttt{[}N\texttt{]} \equiv \texttt{qbit}\ x\texttt{[0]; }\ldots\texttt{; qbit}\ x\texttt{[}N-1\texttt{]}$

$b\ \texttt{:= measure}\ q \equiv \texttt{measure}\ q\ \texttt{then}\ b\ \texttt{:= 1 else}\ b\ \texttt{:= 0}$

$b\texttt{[] := measure}\ q\texttt{[]} \equiv b\texttt{[0] := measure}\ q\texttt{[0]; }\ldots\texttt{; }b\texttt{[}N-1\texttt{] := measure}\ q\texttt{[}N-1\texttt{]}$

$b\texttt{[] :=}\ n \in \mathbb{N} \equiv b\texttt{[0] :=}\ n_{2,0}\texttt{; }\ldots\texttt{; }b\texttt{[}N-1\texttt{] :=}\ n_{2,N-1}$

By the notation $n_{2,i}$ we mean the ith bit of n in base 2, starting from the least significant bit. Note that the `x[i]` notation has no special meaning: it is one literal name.

5.2 Semantics

The semantics also includes typing judgements, which decide the shape of the state in the semantics. We skip these typing judgements to simplify later proofs. The state of a qbit is encoded as a density matrix in $\mathbb{C}^{2\times 2}$. A bit is a *diagonal* density matrix in $\mathbb{C}^{2\times 2}$. The state of the whole system is then the tensor product over the states of all bits and qbits. The order of the variables is determined by the typing context, but since we ignore that, we assume that there is some implied order. Additionally, the semantics assumes that the state that statements relate to are in front, so we assume A is permuted as needed. The semantics is then given as such:

$$\begin{aligned}
N &= |0\rangle\langle 1| + |1\rangle\langle 0| \\
\pi_i(A) &= (E_i \otimes I)A(E_i \otimes I) \\
\nu(A) &= (N \otimes I)A(N \otimes I) \\
[\![\texttt{skip}]\!](A) &= A \\
[\![P\texttt{;}\ Q]\!](A) &= [\![Q]\!]([\![P]\!](A)) \\
[\![\texttt{bit}\ b]\!](A) &= E_0 \otimes A \\
[\![\texttt{qbit}\ q]\!](A) &= E_0 \otimes A \\
[\![\texttt{discard}\ x]\!](A) &= (e_0^\dagger \otimes I)A(e_0 \otimes I) + (e_1^\dagger \otimes I)A(e_1 \otimes I) \\
[\![b\ \texttt{:=}\ \texttt{0}]\!](A) &= \pi_0(A) + \nu(\pi_1(A)) \\
[\![b\ \texttt{:=}\ \texttt{1}]\!](A) &= \nu(\pi_0(A)) + \pi_1(A) \\
[\![x, \ldots, x\ \texttt{*=}\ U]\!](A) &= (U \otimes I)A(U^\dagger \otimes I) \\
[\![\texttt{if}\ b\ \texttt{then}\ P_1\ \texttt{else}\ P_0]\!](A) &= [\![P_0]\!](\pi_0(A)) + [\![P_1]\!](\pi_1(A)) \\
[\![\texttt{while}\ b\ \texttt{do}\ P]\!](A) &= \sum_{n=0}^{\infty} \pi_0(([\![P]\!] \circ \pi_1)^n(A)) \\
[\![\texttt{measure}\ q\ \texttt{then}\ P_1\ \texttt{else}\ P_0]\!](A) &= [\![P_0]\!](\pi_0(A)) + [\![P_1]\!](\pi_1(A))
\end{aligned}$$

5.3 Formulae

Formulae are interpreted with a partial density operator A. The satisfaction relation of a formula is denoted as $A \models \phi$. The $(\models)$ operator also denotes semantic consequence between formulae as $\phi \models \psi$. We additionally chain this operator: $\phi \models \phi' \models \phi''$ is understood to mean $\phi \models \phi'$, $\phi' \models \phi''$. The satisfaction relation is defined as such:

$$\begin{aligned}
&A \models t_1 \leq t_2 \text{ iff } t_1^\circ \leq t_1^\circ \\
&A \models \text{int}(t) \text{ iff } t^\circ \in \mathbb{Z} \\
&A \models t\phi \text{ iff } \exists A' : A = t^\circ A' \wedge A' \models \phi \\
&A \models \phi_1 \oplus \phi_2 \text{ iff } \exists A_1, A_2 : A = A_1 + A_2 \wedge A_1 \models \phi_1 \wedge A_2 \models \phi_2 \\
&A \models {}^r\!M\phi \text{ iff } \exists A' : A = {}^r\!MA'{}^r\!M^\dagger \wedge A' \models \phi \\
&A \models \neg\phi \text{ iff } \neg(A \models \phi) \\
&A \models \phi_1 \wedge \phi_2 \text{ iff } (A \models \phi_1) \wedge (A \models \phi_2) \\
&A \models \forall\alpha : \phi \text{ iff } \forall\beta \in \mathbb{R} : A \models \phi[\beta/\alpha]
\end{aligned} \tag{5.1}$$

A term may contain constants, function applications and a special operator $\text{pr}(\rho)$ that evaluates the probability of ρ holding true. The interpretation of a term t is denoted as t°. We also use the notation $t^{\circ A}$ if the model used for interpretation (A in that case) is ambiguous.

$$\begin{aligned} r^\circ &\equiv r \\ f(t_1, \ldots, t_n)^\circ &\equiv f(t_1^\circ, \ldots, t_n^\circ) \\ \text{pr}(\rho)^\circ &\equiv \sum \{u^\dagger A u \,|\, i_1, \ldots, i_n \in \{0, 1\} \\ &\qquad\qquad\qquad \wedge\, u = e_{i_1} \otimes \ldots \otimes e_{i_n} \\ &\qquad\qquad\qquad \wedge\, \rho[i_1/x_1, \ldots, i_n/x_n] \text{ is true}\} \end{aligned}$$

This definition is slightly altered with respect to [9]. The original model for a formula consists of a density operator A, typing context Γ and binding of quantified and free variables v. The typing context is mostly irrelevant for $(\models)$, except for the transformation rule ${}^r M\phi$, where the typing context Γ is used to permute A. In this research we instead assume that ${}^r M$ denotes the correct extension of M for A instead. Whereas bindings introduced by $(\forall)$ originally are put in v, this research uses substitution instead of adding to v, as in equation (5.1).

We also use one definition as syntactic sugar:

$$[\rho] \equiv \text{pr}(\rho) = \text{pr}()$$

5.4 Proof Rules

The following proof rules are used throughout the proof. Sequential composition is standard:

$$\frac{\{\phi\}\ \texttt{P}\ \{\gamma\} \qquad \{\gamma\}\ \texttt{Q}\ \{\psi\}}{\{\phi\}\ \texttt{P; Q}\ \{\psi\}}\ \text{SEQ}$$

Declaring a bit or qbit initializes it to zero:

$$\frac{}{\{(pr() = 1) \wedge \phi\}\ \texttt{bit}\ b\ \{(pr(\overline{b} = 0) = 1) \wedge \phi\}}\ \text{NEW-BIT}$$

$$\frac{}{\{(pr() = 1) \wedge \phi\}\ \texttt{bit}\ q\ \{(pr(\overline{q} = 0) = 1) \wedge \phi\}}\ \text{NEW-QBIT}$$

Assigning a constant to a bit or qbit transforms the system state to only that outcome. Intuitively ${}^b E_0 \phi$ filters for the situation where $b = 0$, whereas ${}^b N E_1 \phi$ filters for the situation $b = 1$, and then inverts b.

$$\frac{}{\{\phi\}\ b\ \texttt{:= 0}\ \{{}^b E_0 \phi \oplus {}^b N E_1 \phi\}}\ \text{ASSIGN}_0$$

$$\frac{}{\{\phi\}\ b\ \texttt{:= 1}\ \{{}^b N E_0 \phi \oplus {}^b E_1 \phi\}}\ \text{ASSIGN}_1$$

Unitary transformation transforms the system with U:

$$\frac{}{\{{}^{\overrightarrow{q}} U^\dagger \phi\}\ \overrightarrow{q}\ \texttt{*=}\ U\ \{\phi\}}\ \text{UNITARY}$$

Alternatively, U may equivalently appear in the postcondition instead:

$$\frac{}{\{\phi\}\ \overrightarrow{q}\ \texttt{*=}\ U\ \{{}^{\overrightarrow{q}}U\phi\}}\ \textsc{unitary}$$

`if` is modeled as a probabilistic branch. In the negative case we filter for $b = 0$ and obtain postcondition ψ_0. In the positive case the opposite holds. ψ_0 and ψ_1 are then joined again with probabilistic sum.

$$\frac{\{{}^{b}E_0\phi\}\ P_0\ \{\psi_0\} \qquad \{{}^{b}E_1\phi\}\ P_1\ \{\psi_1\}}{\{\phi\}\ \texttt{if}\ b\ \texttt{then}\ P_1\ \texttt{else}\ P_0\ \{\psi_0 \oplus \psi_1\}}\ \textsc{if}$$

The rule for `while` allows for a predicate to be parametric on the number of iterations:

$$\frac{\{{}^{b}E_1\phi_n\}\ P\ \{\phi_{n+1}\}\ (n \in \mathbb{N}) \qquad \{{}^{b}E_0\phi_n \mid n \in \mathbb{N}\} \models \psi}{\{\phi_0\}\ \texttt{while}\ b\ \texttt{do P}\ \{\psi\}}\ \textsc{while}$$

If all ϕ_i are instead equal, the rule becomes the standard rule for `while` with a loop invariant. This is the variant we use:

$$\frac{\{{}^{b}E_1\phi\}\ P\ \{\phi\}}{\{\phi\}\ \texttt{while}\ b\ \texttt{do P}\ \{{}^{b}E_0\phi\}}\ \textsc{while}$$

`measure` is the equivalent of `if` for qbits and is otherwise identical to the IF rule.

$$\frac{\{{}^{q}E_0\phi\}\ P_0\ \{\psi_0\} \qquad \{{}^{q}E_1\phi\}\ P_1\ \{\psi_1\}}{\{\phi\}\ \texttt{measure}\ q\ \texttt{then}\ P_1\ \texttt{else}\ P_0\ \{\psi_0 \oplus \psi_1\}}\ \textsc{measure}$$

We may apply standard logic to strengthen the precondition, or weaken the postcondition:

$$\frac{\phi \models \phi' \qquad \{\phi'\}\ P\ \{\psi'\} \qquad \psi' \models \psi}{\{\phi\}\ P\ \{\psi\}}\ \textsc{logic}$$

Chapter 6

Shor's Factoring Algorithm

Shor's factoring algorithm [10] produces, as the name suggests, a non-trivial factor of a composite number. The speed-up as compared to the fastest known classical algorithm is enormous. With n the number of bits of the number to be factored the classical algorithm is in $\tilde{O}(2^{n/4+o(1)})$, whereas the quantum algorithm is in $\tilde{O}(n^3)$ [8]. For the algorithm to work we first must rule out some situations we can deal with efficiently with classical computing. This makes it a good case to evaluate both the ability of a logic to deal with quantum concepts, as well as the interaction with classical code.

Below is an introduction to how the algorithm works. The description is largely taken from [12], though we will not reproduce some parts of the motivation of steps of the algorithm here.

The algorithm starts by reducing factor-finding to the problem of finding the period of a function. Suppose we are trying to decompose N. We may assume that N **is not even**, and N **is not the power of a prime** ($\nexists i > 1, p \text{ prime} : N = p^i$), since we can efficiently rule out those situations with classical computing.

We randomly choose an integer $x \in [2, N)$. We may assume **x is coprime with** N, since otherwise we have already found a non-trivial factor accidentally and we are done. The function we will consider the period of is:

$$f(a) = x^a \mod N$$

It follows that $f(0) = 1$. Since the function can have at most N distinct outcomes, there must be a minimal $0 < r \leq N$ such that $f(0) = f(r) = 1$. Since $f(a+b) = f(a) \cdot f(b) \mod N$, we can see that $f(a) = f(a+r) = f(a+2r) = \ldots$ Therefore we call r the period of the function f.

We can now use r (for which we have not yet given an algorithm) to find a non-trivial factor of N due to the following equivalencies:

$$\begin{aligned} x^r &= 1 \mod N \\ (x^{r/2})^2 &= 1 \mod N \\ (x^{r/2}+1)(x^{r/2}-1) &= 0 \mod N \end{aligned}$$

It can be shown that with probability greater than 1/2 that r is even, and neither $(x^{r/2}+1)$ nor $(x^{r/2}-1)$ are multiples of N. A constant non-zero probability is sufficient, since we only need to repeat the algorithm a constant number of times to achieve a desired likelihood of finding a factor. Since neither quantity $(x^{r/2}+1)$ nor $(x^{r/2}-1)$ contains all factors of N, but both quantities together contain all factors of N, they both share a non-trivial factor with N. These factors then are $gcd(x^{r/2}+1, N)$ and $gcd(x^{r/2}-1, N)$, which are easy to compute classically.

This concludes the reduction of factor-finding to period-finding. Here follows a description of Shor's period-finding algorithm.

The algorithm starts with a quantum register a of size $l = \lfloor \log N^2 \rfloor + 1$ and a quantum register f_a of size $n = \lceil \log N \rceil$. For convenience we introduce $q = 2^l$ to count over the possible values of a. We initialize the first register a to have uniform likelihood over all measurement outcomes. The second register f_a is initialized to 0. The algorithm also requires a black-box operator O_f that transforms $|a\rangle|0\rangle$ to $|a\rangle|f(a)\rangle$.

The system starts in superposition:

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|0\rangle$$

We then apply the black-box operator O_f:

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|f(a)\rangle$$

We take a measurement of the f_a register, which we denote with $f(s)$ where $s < r$. Of course $f(s) = f(s+r) = f(s+2r) = \ldots$, so there are either $m = \lceil q/r \rceil$ or $m = \lfloor q/r \rfloor$ remaining possible outcomes for a. The f_a register is in the classical state $f(s)$ due to the measurement, so we only consider the a register from here. This register is now in the superposition:

$$\frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} |s + jr\rangle$$

We apply the quantum Fourier transform (QFT) to a, leaving us with:

$$\frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} \frac{1}{\sqrt{q}} \sum_{b=0}^{q-1} e^{2\pi i \frac{(s+jr)b}{q}} |b\rangle \tag{6.1}$$

$$= \frac{1}{\sqrt{mq}} \sum_{b=0}^{q-1} e^{2\pi i sb/q} \left[\sum_{j=0}^{m-1} (e^{2\pi i rb/q})^j \right] |b\rangle$$

We are now interested in which measurements of a are likely. That means we must determine which $|b\rangle$ have a high probability amplitude before them. The expression $(e^{2\pi i rb/q})^j$ denotes a geometric series, so we can simplify:

$$\sum_{j=0}^{m-1} (e^{2\pi i rb/q})^j = \begin{cases} m & \text{when } e^{2\pi i rb/q} = 0 \quad (6.2) \\ \dfrac{1 - e^{2\pi i mrb/q}}{1 - e^{2\pi i rb/q}} & \text{when } e^{2\pi i rb/q} \neq 0 \quad (6.3) \end{cases}$$

Case (6.2) is vanishingly unlikely, so we will only consider the amplitude of case (6.3):

$$\left|\frac{1 - e^{2\pi imrb/q}}{1 - e^{2\pi irb/q}}\right| = \frac{|1 - e^{2\pi imrb/q}|}{|1 - e^{2\pi irb/q}|} = \frac{|\sin \pi mrb/q|}{|\sin \pi rb/q|} \tag{6.4}$$

The value in (6.4) is high when the denominator $|\sin \pi rb/q|$ is small, which is the case when b is close to a multiple of q/r: rb/q is then close to an integer. For most such b, the numerator is not close to 0, because usually rb/q is not exactly an integer and $r/bq.$ is multiplied with m in the numerator.

Finally, we measure a, giving us some measurement outcome b as above. With high likelihood there is a c such that b is close to cq/r, or equivalently b/q is close to c/r. In fact, with high probability we can find the unique $c, r \in \mathbb{N}$ such that:

$$\left|\frac{b}{q} - \frac{c}{r}\right| \leq \frac{1}{2q}$$

The fraction c/r can be obtained using continued fraction expansion, a procedure than can be done efficiently classically [3]. This will not be further explained here.

6.1 Pseudocode

We give an implementation of the factoring algorithm in pseudocode, so we can show a correspondence between this implementation and the implementation in the quantum language of the logic in Chapter 5. The program is given in Algorithm 6.1.

```
if $N$ is even then
    return 2
else
    if $N$ is a power of prime $p$ then
        return $p$
    else
        loop
            $x \leftarrow \mathsf{sample}\ [2, N)$
            $a \leftarrow 0$
            $f_a \leftarrow 0$
            $a \leftarrow \mathsf{QFT}(a)$
            $a : f_a \leftarrow \mathsf{O_f}(a : f_a)$
            $\_ \leftarrow \mathsf{measure}\ f_a$
            $a \leftarrow \mathsf{QFT}(a)$
            $b \leftarrow \mathsf{measure}\ a$
            Apply continued fraction expansion to $b/q$ yielding $c/r$
            if $1 < \mathsf{gcd}(x^{r/2} + 1, N) < N$ then
                return $\mathsf{gcd}(x^{r/2} + 1, N)$
            else if $1 < \mathsf{gcd}(x^{r/2} - 1, N) < N$ then
                return $\mathsf{gcd}(x^{r/2} - 1, N)$
            end if
        end loop
    end if
end if
```

Algorithm 1: Pseudocode implementation of Shor's algorithm

Chapter 7

Lemmas

In this chapter we introduce lemmas that are necessary for the case study in Chapter 8. The lemmas are broadly divided into logic-level lemmas that prove equivalent formulae (as defined in Section 5.3), extra proof rules (similar to the ones in Section 5.4), and mathematical lemmas.

7.1 Logic-Level Lemmas

We start by introducing some lemmas that state equivalences between predicates. These are used in later extra proof rules, and with the LOGIC rule.

7.1.1 Probability is not Greater Than One

As described above, $\mathrm{pr}(\rho)$ measures a probability. We show that $\mathrm{pr}(\rho)$ does not evaluate to more than $\mathrm{pr}()$. Furthermore A is a partial density matrix, so we can state that $\mathrm{pr}(\rho)$ never evaluates to more than 1.

Lemma 1. *$pr(\rho)^\circ \leq pr()^\circ \leq 1$*

Proof. Both inequalities are shown by expanding the definition of $\mathrm{pr}(\rho)^\circ$:

$$\begin{aligned}
\mathrm{pr}(\rho)^\circ &= \sum\{u^\dagger A u \mid i_1, \ldots, i_n \in \{0,1\} \\
&\qquad\qquad \wedge\, u = e_{i_1} \otimes \ldots \otimes e_{i_n} \\
&\qquad\qquad \wedge\, \rho[i_1/x_1, \ldots, i_n/x_n] \text{ is true}\} \\
&\leq \sum\{u^\dagger A u \mid i_1, \ldots, i_n \in \{0,1\} \\
&\qquad\qquad \wedge\, u = e_{i_1} \otimes \ldots \otimes e_{i_n}\} \\
&= \mathrm{pr}()^\circ \\
&= \mathrm{tr}(A) \leq 1
\end{aligned}$$

□

7.1.2 Measurement Outcome in Probability Predicate

It is often useful to use a measurement outcome in a probability predicate $\mathrm{pr}(\rho)$. This can be done as follows:

Lemma 2. *For $i \in \{0,1\}$: ${}^rE_i pr(\rho) = pr() \models {}^rE_i pr(\rho \wedge r = i) = pr()$*

Proof. By definition the relation above is true when for any partial density operator A: $A \models {}^rE_i\mathrm{pr}(\rho) = \mathrm{pr}()$ implies $A \models {}^rE_i\mathrm{pr}(\rho \wedge r = i) = \mathrm{pr}()$.

Assuming $A \models {}^rE_i\mathrm{pr}(\rho) = \mathrm{pr}()$, we have that there is A' such that $A = {}^rE_i A' {}^rE_i$ and $A' \models \mathrm{pr}(\rho) = \mathrm{pr}()$. Furthermore, since ${}^rE_i A {}^rE_i = {}^rE_i A' {}^rE_i$, we also have $A = {}^rE_i A {}^rE_i$. It is then sufficient to show that $A \models \mathrm{pr}(\rho \wedge r = i) = \mathrm{pr}()$.

We now interpret the remaining formula $\mathrm{pr}(\rho \wedge r = i) = \mathrm{pr}()$ with the model A, unfolding the definition of $\mathrm{pr}(\rho)$:

$$\begin{aligned}\mathrm{pr}(\rho \wedge r = i)^\circ = \sum\{u^\dagger A u \,|\, & i_1, \ldots, i_n \in \{0,1\}, \\ & (\rho \wedge r = i)[r_1/i_1, \ldots, r_n/i_n] \text{ is true}, \\ & u = e_{i_1} \otimes \cdots \otimes e_{i_n}\} = \mathrm{pr}()^\circ\end{aligned}$$

Since A is a density matrix, we know $u^\dagger A u \geq 0$. We continue with a proof by contradiction, noting that if the equality above is not true, there must be some $u^\dagger A u > 0$ and $(\rho \wedge r = i)$ false under interpretation with $i_1, \ldots, i_n$.

If we have $i_r = \bar{i}$, we can compute:

$$\begin{aligned}u^\dagger A u &= u^{\dagger r}E_i A' {}^rE_i u \\ &= u^{\dagger r}E_i A' {}^rE_i(e_{i_1} \otimes \ldots \otimes e_{i_r} \otimes \ldots \otimes e_{i_n}) \\ &= u^{\dagger r}E_i A' {}^rE_i(e_{i_1} \otimes \ldots \otimes e_{\bar{i}} \otimes \ldots \otimes e_{i_n}) \\ &= u^{\dagger r}E_i A' 0 \\ &= 0\end{aligned}$$

This contradicts $u^\dagger A u > 0$. If we have on the other hand that $i_r = i$, we can derive from $\mathrm{pr}(\rho)^\circ = \mathrm{pr}()^\circ$ under interpretation with A' that either ρ is true under interpretation with $i_1, \ldots, i_n$, or $u^\dagger A' u = 0$. Clearly we then also have $(\rho \wedge r = i)$ is true, or $u^{\dagger r}E_i A' {}^rE_i u = u^\dagger A u = 0$: a contradiction.

We therefore conclude that $\mathrm{pr}(\rho \wedge r = i)^\circ = \mathrm{pr}()^\circ$ under A, and hence the lemma holds. □

7.1.3 Diagonal Formulae

We start by defining *diagonal formulae*. Diagonal formulae are a regular formula in the logic, that are restricted in all occurences of the form $M\phi$: M must be diagonal.[1] For example, $N\phi$ is not diagonal, since N is not a diagonal matrix. As another example, $E_0\phi \oplus E_1\psi$ is diagonal iff ϕ and ψ are diagonal. For completeness sake, we define the diagonal of a matrix M as the column vector:

$$\mathsf{diag}(M)_i = M_{ii}$$

We prove that diagonal formulae only depends on the diagonal of an interpretation A. First we define $t^{\circ D}$ (interpretation of term t with D) and $D \models \phi$,

[1]A diagonal matrix is zero in all off-diagonal entries.

where D is the vector that represents the diagonal of a matrix. Then we show that $A \models \phi$ iff $\mathsf{diag}(A) \models \phi$.

$$\begin{aligned}
r^{\circ D} &= r \\
(f(t_1,\ldots,t_n))^{\circ D} &= f(t_1^{\circ D},\ldots,t_n^{\circ D}) \\
(\mathrm{pr}(\rho))^{\circ D} &= \sum\{u^\dagger D \,|\, i_1,\dots \in \{0,1\} \wedge u = e_{i_1} \otimes \dots \wedge \rho[i_1/r_1,\dots] \text{ is true}\}
\end{aligned}$$

$$\begin{aligned}
&D \models t_1 \leq t_2 \text{ iff } t_1^{\circ D} \leq t_2^{\circ D} \\
&D \models \mathrm{int}(t) \text{ iff } t^{\circ D} \in \mathbb{Z} \\
&D \models t\phi \text{ iff } \exists D' : D = t^{\circ D} D' \wedge D' \models \phi \\
&D \models \phi_1 \oplus \phi_2 \text{ iff } \exists D_1, D_2 : D = D_1 + D_2 \wedge D_1 \models \phi_1 \wedge D_2 \models \phi_2 \\
&D \models {}^r\!M\phi \text{ iff } \exists D' : D = {}^r\!M D' {}^r\!M^\dagger \wedge D' \models \phi \\
&D \models \neg\phi \text{ iff } \neg(D \models \phi) \\
&D \models \phi_1 \wedge \phi_2 \text{ iff } (D \models \phi_1) \wedge (D \models \phi_2) \\
&D \models \forall\alpha : \phi \text{ iff } \forall\beta \in \mathbb{R} : D \models \phi[\beta/\alpha]
\end{aligned}$$

We can show with simple algebra that $t^{\circ A}$ is equal to $t^{\circ \mathsf{diag}(A)}$. Consequently, t° may interchangeably mean interpretation with $\mathsf{diag}(A)$ or A below. Left to prove is then that for diagonal formulae $A \models \phi$ if and only if $\mathsf{diag}(A) \models \phi$:

Lemma 3. *For diagonal ϕ: $A \models \phi$ iff $\mathsf{diag}(A) \models \phi$*

Proof. The proof is by complete structural induction on ϕ. That is: for every way to construct a formula and assuming that all smaller formulae obey our desired property, we prove the new larger formula has that property. We may then conclude that all formulae have the desired property.

- $A \models t_1 \leq t_2$ iff $\mathsf{diag}(A) \models t_1 \leq t_2$: Terms are equal under interpretation with A and $\mathsf{diag}(A)$.
- $A \models \mathrm{int}(t)$ iff $\mathsf{diag}(A) \models \mathrm{int}(t)$: The same argument holds.
- $A \models t\phi$ iff $\mathsf{diag}(A) \models t\phi$: By definition, we have to show $\exists D' : \mathsf{diag}(A) = t^{\circ\mathsf{diag}(A)} D' \wedge D' \models \phi$ iff $\exists A' : A = t^{\circ A} A' \wedge A' \models \phi$. We can see that $\mathsf{diag}(A) = \mathsf{diag}(t^{\circ A} A') = t^{\circ A}\mathsf{diag}(A')$. If $t^{\circ A} = 0$ the result follows trivially. Otherwise $D' = \mathsf{diag}(A')$ and by the induction hypothesis $A' \models \phi$ iff $\mathsf{diag}(A') \models \phi$.
- $A \models \phi_1 \oplus \phi_2$ iff $\mathsf{diag}(A) \models \phi_1 \oplus \phi_2$

  Expanding the definition we must show:

  $$\begin{aligned}
  &\exists A_1, A_2 : A = A_1 + A_2 \wedge A_1 \models \phi_1 \wedge A_2 \models \phi_2 \\
  \text{iff } &\exists D_1, D_2 : \mathsf{diag}(A) = D_1 + D_2 \wedge D_1 \models \phi_1 \wedge D_2 \models \phi_2
  \end{aligned}$$

  ($\Rightarrow$) If we obtain such A_1, A_2 we may set $D_1 = \mathsf{diag}(A_1)$, $D_2 = \mathsf{diag}(A_2)$. By the induction hypothesis $D_1 \models \phi_1$ and $D_2 \models \phi_2$.

  ($\Leftarrow$) Similarly from D_1, D_2 we can set $A_1 = \mathsf{diag}^{-1}(D_1)$, $A_2 = A - A_1$.[2] We have $\mathsf{diag}(A_1) = \mathsf{diag}(\mathsf{diag}^{-1}(D_1)) = D_1$ and $\mathsf{diag}(A_2) = \mathsf{diag}(A - A_1) = \mathsf{diag}(A) - \mathsf{diag}(A_1) = D_2$. By the induction hypothesis we have $A_1 \models \phi_1$ and $A_2 \models \phi_2$.

[2]We take $\mathsf{diag}^{-1}(D)$ to mean the diagonal matrix with diagonal D

- $A \models {}^r\!M\phi$ iff $\mathsf{diag}(A) \models {}^r\!M\phi$

  By definition this equates to:

$$\begin{aligned} &\exists A' : A = {}^r\!MA'M^\dagger \wedge A' \models \phi \\ \text{iff } &\exists D' : \mathsf{diag}(A) = {}^r\!MD'M^\dagger \wedge D' \models \phi \end{aligned}$$

  ($\Rightarrow$) We have A' such that $A = {}^r\!MA'M^\dagger$ and $A' \models \phi$. Since M is diagonal, we can establish that $\mathsf{diag}({}^r\!MA'{}^r\!M^\dagger) = {}^r\!M\mathsf{diag}(A'){}^r\!M^\dagger$. By assigning $D' = \mathsf{diag}(A')$ we have:

$$\begin{aligned} \mathsf{diag}(A) &= \mathsf{diag}({}^r\!MA'M^\dagger) \\ &= {}^r\!M\mathsf{diag}(A')M^\dagger \\ &= {}^r\!MD'M^\dagger \end{aligned}$$

  From the induction hypothesis we have $A' \models \phi \Rightarrow D' \models \phi$, so we are done.

  ($\Leftarrow$) We have D' such that $\mathsf{diag}(A) = {}^r\!MD'M^\dagger$ and $D' \models \phi$. Due to the fact that M is diagonal, we have $A_{ij} = M_{ii}A'_{ij}M^\dagger_{jj}$. We can therefore set the diagonal of A' to D', which will be sufficient to satisfy $A' \models \phi$ due to the induction hypothesis. The off-diagonals must satisfy $A = {}^r\!MA'M^\dagger$:

$$A'_{ij} = \begin{cases} D'_i & \text{if } i = j \\ 0 & \text{if } M_{ii} = 0 \vee M_{jj} = 0 \\ A_{ij}/(M_{ii}M^\dagger_{jj}) & \text{otherwise} \end{cases}$$

- $A \models \neg\phi$ iff $\mathsf{diag}(A) \models \neg\phi$: Expanding the definition we get the induction hypothesis: $\neg(A \models \phi)$ iff $\neg(\mathsf{diag}(A) \models \phi)$.

- $A \models \phi_1 \wedge \phi_2$ iff $\mathsf{diag}(A) \models \phi_1 \wedge \phi_2$

  From the definition we must prove: $A \models \phi_1 \wedge A \models \phi_2$ iff $\mathsf{diag}(A) \models \phi_1 \wedge \mathsf{diag}(A) \models \phi_2$, which follows from the induction hypothesis.

- $A \models \forall\alpha : \phi$ iff $\mathsf{diag}(A) \models \forall\alpha : \phi$

  Again from the definition this is equivalent to: $(\forall\beta \in \mathbb{R} : A \models \phi[\beta/\alpha])$ iff $(\forall\beta \in \mathbb{R} : \mathsf{diag}(A) \models \phi[\beta/\alpha])$. For any β we have $A \models \phi[\beta/\alpha]$ iff $\mathsf{diag}(A) \models \phi[\beta/\alpha]$, since substitution with a constant leaves us with a smaller formula.

We can conclude that for diagonal ϕ: $A \models \phi$ iff $\mathsf{diag}(A) \models \phi$. □

7.1.4 Distribute over E_0 and E_1

We show that $\text{pr}(\rho) = \text{pr}()$ is equivalent to $E_0(\text{pr}(\rho) = \text{pr}()) \oplus E_1(\text{pr}(\rho) = \text{pr}())$:

Lemma 4. *$pr(\rho) = pr() \models, \Dashv E_0(pr(\rho) = pr()) \oplus E_1(pr(\rho) = pr())$*

Proof. By Lemma 3 we may prove instead:

$$D \models \text{pr}(\rho) = \text{pr}() \text{ iff } D \models E_0(\text{pr}(\rho) = \text{pr}()) \oplus E_1(\text{pr}(\rho) = \text{pr}())$$

($\Rightarrow$) Set $D_0 = E_0DE_0$, $D_1 = E_1DE_1$, $D'_0 = D_0$, $D'_1 = D_1$. Clearly $D = D_0 + D_1$, $D_0 = E_0D'_0E_0$ and $D_1 = E_1D'_1E_1$. Then $D'_0 \models \text{pr}(\rho) = \text{pr}()$ and $D'_1 \models \text{pr}(\rho) = \text{pr}()$.

($\Leftarrow$) We have D_0, D_1 such that $D = D_0 + D_1$, $D_0 \models E_0(\mathrm{pr}(\rho) = \mathrm{pr}())$ and $D_1 \models E_1(\mathrm{pr}(\rho) = \mathrm{pr}())$. By Lemma 2 we have $D_0 \models E_0(\mathrm{pr}(\rho \wedge r = 0) = \mathrm{pr}())$ and $D_1 \models E_1(\mathrm{pr}(\rho \wedge r = 1) = \mathrm{pr}())$. We then obtain D'_0, D'_1 such that $D_0 = E_0 D'_0 E_0$, $D_1 = E_1 D'_1 E_1$, $D'_0 \models \mathrm{pr}(\rho \wedge r = 0) = \mathrm{pr}()$ and $D'_1 \models \mathrm{pr}(\rho \wedge r = 1) = \mathrm{pr}()$. Then:

$$\begin{aligned}
\mathrm{pr}(\rho \wedge r = 0)^{\circ D'_0} &= \sum\{u^\dagger D'_0 \,|\, i_1, \ldots, i_n \in \{0,1\} \\
&\qquad \wedge u = e_{i_1} \otimes \cdots \otimes e_{i_n} \\
&\qquad \wedge (\rho \wedge r = 0)[i_1/r_1, \ldots, i_n/r_n] \text{ is true}\} \\
&= \sum\{u^\dagger D'_0 \,|\, i_1, \ldots, i_n \in \{0,1\} \wedge i_r = 0 \\
&\qquad \wedge u = e_{i_1} \otimes \cdots \otimes e_{i_n} \\
&\qquad \wedge \rho[i_1/r_1, \ldots, i_n/r_n] \text{ is true}\} \\
&= \sum\{u^\dagger E_0 D'_0 E_0 \,|\, i_1, \ldots, i_n \in \{0,1\} \\
&\qquad \wedge u = e_{i_1} \otimes \cdots \otimes e_{i_n} \\
&\qquad \wedge \rho[i_1/r_1, \ldots, i_n/r_n] \text{ is true}\} \\
&= \sum\{u^\dagger D_0 \,|\, i_1, \ldots, i_n \in \{0,1\} \\
&\qquad \wedge u = e_{i_1} \otimes \cdots \otimes e_{i_n} \\
&\qquad \wedge \rho[i_1/r_1, \ldots, i_n/r_n] \text{ is true}\} \\
&= \mathrm{pr}(\rho)^{\circ D_0} \\
\mathrm{pr}()^{\circ D_0} &= \sum\{u^\dagger D_0 \,|\, i_1, \ldots, i_n \in \{0,1\} \wedge u = e_{i_1} \otimes \cdots \otimes e_{i_n}\} \\
&= \sum\{u^\dagger E_0 D'_0 E_0 \,|\, i_1, \ldots, i_n \in \{0,1\} \wedge u = e_{i_1} \otimes \cdots \otimes e_{i_n}\} \\
&= \sum\{u^\dagger D'_0 \,|\, i_1, \ldots, i_n \in \{0,1\} \wedge u = e_{i_1} \otimes \cdots \otimes e_{i_n}\} \\
&= \mathrm{pr}()^{\circ D'_0}
\end{aligned}$$

Similarly $\mathrm{pr}(\rho \wedge r = 1)^{\circ D'_1} = \mathrm{pr}(\rho)^{\circ D_1}$ and $\mathrm{pr}()^{\circ D_1} = \mathrm{pr}()^{\circ D'_1}$. Since $D = D_0 + D_1$, we have:

$$\begin{aligned}
\mathrm{pr}(\rho)^{\circ D} &= \mathrm{pr}(\rho)^{\circ D_0} + \mathrm{pr}(\rho)^{\circ D_1} \\
&= \mathrm{pr}(\rho \wedge r = 0)^{\circ D'_0} + \mathrm{pr}(\rho \wedge r = 1)^{\circ D'_1} \\
&= \mathrm{pr}()^{\circ D'_0} + \mathrm{pr}()^{\circ D'_1} \\
&= \mathrm{pr}()^{\circ D_0} + \mathrm{pr}()^{\circ D_1} \\
&= \mathrm{pr}()^{\circ D}
\end{aligned}$$

We can therefore conclude that $D \models \mathrm{pr}(\rho) = \mathrm{pr}()$.

$\square$

7.1.5 Independent Measurement

We show that measurement of two different bits r and q is commutative:

Lemma 5. $r \neq q \Rightarrow (A \models {}^rE_i{}^qE_j\phi$ iff $A \models {}^qE_j{}^rE_i\phi)$

Proof. Noting that the matrices ${}^{r}E_i$ and ${}^{q}E_j$ are already commutative we interpret the formula with A:

$$\begin{aligned}
&A \models {}^{r}E_i{}^{q}E_j\phi\\
\text{iff }& \exists A' : A = {}^{r}E_iA'{}^{r}E_i \wedge A' \models {}^{q}E_j\phi\\
\text{iff }& \exists A' : A = {}^{r}E_iA'{}^{r}E_i \wedge \exists A'' : A' = {}^{q}E_jA''{}^{q}E_j \wedge A'' \models \phi\\
\text{iff }& \exists A', A'' : A = {}^{r}E_iA'{}^{r}E_i \wedge A' = {}^{q}E_jA''{}^{q}E_j \wedge A'' \models \phi\\
\text{iff }& \exists A'' : A = {}^{r}E_i{}^{q}E_jA''{}^{q}E_j{}^{r}E_i \wedge A'' \models \phi \wedge \exists A' : A' = {}^{q}E_jA''{}^{q}E_j\\
\text{iff }& \exists A'' : A = {}^{r}E_i{}^{q}E_jA''{}^{q}E_j{}^{r}E_i \wedge A'' \models \phi\\
\text{iff }& \exists A'' : A = {}^{q}E_j{}^{r}E_iA''{}^{r}E_i{}^{q}E_j \wedge A'' \models \phi\\
\text{iff }& A \models {}^{q}E_j{}^{r}E_i\phi
\end{aligned}$$

□

7.1.6 Distribute Unitary Operations over $\oplus$

We show that unitary operations distribute over probabilistic sum, that is:

Lemma 6. $A \models M(\phi \oplus \psi)$ iff $A \models M\phi \oplus M\psi$

Proof. This is established from the fact that the matrix M distributes over addition, unfolding definitions of formulae:

$$\begin{aligned}
&A \models M(\phi \oplus \psi)\\
\text{iff }& \exists A' : A = MA'M^\dagger \wedge A' \models \phi \oplus \psi\\
\text{iff }& \exists A', A'_\phi, A'_\psi : A = MA'M^\dagger \wedge A' = A'_\phi + A'_\psi \wedge A'_\phi \models \phi \wedge A'_\psi \models \psi\\
\text{iff }& \exists A'_\phi, A'_\psi : A = M(A'_\phi + A'_\psi)M^\dagger \wedge A'_\phi \models \phi \wedge A'_\psi \models \psi\\
\text{iff }& \exists A'_\phi, A'_\psi : A = MA'_\phi M^\dagger + MA'_\psi M^\dagger \wedge A'_\phi \models \phi \wedge A'_\psi \models \psi\\
\text{iff }& \exists A_\phi, A_\psi : A = A_\phi + A_\psi \wedge A_\phi \models M\phi \wedge A_\psi \models M\psi\\
\text{iff }& A \models M\phi \oplus M\psi
\end{aligned}$$

□

7.2 Proof Rules

This section contains extra proof rules. They broaden existing proof rules, as well as specify rules for the introduced syntactic sugar introduced in Section 5.1.

7.2.1 Assign

By axioms ASSIGN-0 and ASSIGN-1 we have:

$$\frac{}{\{\phi\}\ \texttt{x := } i\ \{N^{x}E_{\bar{i}}\phi \oplus {}^{x}E_i\phi\}}\ \text{ASSIGN-}i$$

We want to show that:

$$\frac{}{\{[\phi \wedge x = c]\}\ \texttt{x := }\ i\ \{[\phi \wedge x = i]\}}$$

Applying the ASSIGN rule we already have:

$$\{[\phi \wedge x = c]\}\ \texttt{x := }\ i\ \{N{}^xE_{\bar{i}}[\phi \wedge x = c] \oplus {}^xE_i[\phi \wedge x = c]\}$$

We should therefore show that $N{}^xE_{\bar{i}}[\phi \wedge x = c] \oplus {}^xE_i[\phi \wedge x = c] \models [\phi \wedge x = i]$, or: $(A_l \models N{}^xE_{\bar{i}}[\phi \wedge x = c]) \wedge (A_r \models {}^xE_i[\phi \wedge x = c]) \Rightarrow A_l + A_r \models [\phi \wedge x = i]$. We now split by $i = c$ and $i \neq c$:

- $\mathbf{i} = \mathbf{c}$: because $\bar{i} \neq c$ we can derive that $A_l = 0$. For A_r we have $A_r = {}^xE_c A'_r {}^xE_c$ and $A'_r \models [\phi \wedge x = c]$. Since in the formula $x = c$ we can derive ${}^xE_{\bar{c}} A'_r {}^xE_{\bar{c}} = 0$. Therefore $\mathrm{pr}(\phi \wedge x = c)^{\circ A'_r} = \mathrm{pr}(\phi \wedge x = c)^{\circ A_r}$ and $A_r \models [\phi \wedge x = i]$.
- $\mathbf{i} \neq \mathbf{c}$: similarly we can derive $A_r = 0$. For A_l we have $A_l = {}^xN{}^xE_{\bar{c}} A'_l {}^xE_{\bar{c}} {}^xN = {}^xE_c A'_l {}^xE_c$ and $A'_l \models [\phi \wedge x = c]$. Again we have in the formula $x = c$ and hence ${}^xE_{\bar{c}} A'_l {}^xE_{\bar{c}} = 0$, and therefore $\mathrm{pr}(\phi \wedge x = c)^{\circ A'_l} = \mathrm{pr}(\phi \wedge x = c)^{\circ A_l}$ and $A_l \models [\phi \wedge x = i]$.

We conclude that $A_l + A_r \models [\phi \wedge x = i]$ and hence:

$$\frac{}{\{[\phi \wedge x = c]\}\ \texttt{x := }\ i\ \{[\phi \wedge x = i]\}}\ \textsc{assign-pr-}i$$

We continue by proving that the Hoare triple may be framed with transformations unrelated to the assigned variable:

$$\frac{\begin{array}{c}\{\phi\}\ \texttt{x := }\ i\ \{\psi\} \\ M{}^xE_i = {}^xE_iM \qquad M{}^xE_{\bar{i}} = {}^xE_{\bar{i}}M \qquad M{}^xN = {}^xNM\end{array}}{\{M\phi\}\ \texttt{x := }\ i\ \{M\psi\}}\ \textsc{frame-assign-}i$$

Since $\{\phi\}\ \texttt{x := }\ i\ \{\psi\}$ we have that for any $A \models \phi$, $[\texttt{x := }\ i](A) \models \psi$. We want to show that $(A \models M\phi) \Rightarrow ([\texttt{x := }\ i](A) \models M\psi)$. By definition we have $A' \models \phi \wedge A = MA'M^\dagger$, so it is sufficient to show $[\texttt{x := }\ i](MA'M^\dagger) \models M\psi$. We can use that M commutes with ${}^xE_i$, ${}^xE_{\bar{i}}$ and ${}^xN$:

$$\begin{aligned}
[\texttt{x := }\ i](MA'M^\dagger) &= {}^xE_iMA'M^{\dagger x}E_i + {}^xN{}^xE_{\bar{i}}MA'M^{\dagger x}E_{\bar{i}}{}^xN \\
&= M{}^xE_iA'{}^xE_iM^\dagger + M{}^xN{}^xE_{\bar{i}}A'{}^xE_{\bar{i}}{}^xNM^\dagger \\
&= M({}^xE_iA'{}^xE_i + {}^xN{}^xE_{\bar{i}}A'{}^xE_{\bar{i}}{}^xN)M^\dagger \\
&= M[\texttt{x := }\ i](A')M^\dagger
\end{aligned}$$

Since $A' \models \phi$ we have that $[\texttt{x := }\ i](A') \models \psi$ and so we can conclude $M[\texttt{x := }\ i](A')M^\dagger \models M\psi$. From here we abbreviate proof trees that are a repeated application of FRAME-ASSIGN-i to ASSIGN-PR-i to simply ASSIGN.

7.2.2 Repeated Declaration

Repeated declaration accumulates a series of (q)bits that are zero. After `bit` x`[0]` we have $[x[0] = 0]$, then after `bit` x`[1]` we have $[x[0] = 0 \wedge x[1] = 0]$ etc. We can see that the syntactic sugar for repeated declarations follows:

$$\frac{}{\{[\phi]\}\ \texttt{bit}\ x\texttt{[}N\texttt{]}\ \{[\phi \wedge x = 0]\}}\ \textsc{new-bit}$$

$$\frac{}{\{[\phi]\}\ \texttt{qbit}\ x\texttt{[}N\texttt{]}\ \{[\phi \wedge x = 0]\}}\ \textsc{new-qbit}$$

7.2.3 Constant Assignment

In a similar fashion we can repeat assignment setting the bits of a sequence of (q)bits. Using the ASSIGN-PR-i rule, we can define a similar rule for the syntactic sugar for assigning a constant to multiple (q)bits:

$$\frac{}{\{[x = c]\}\ \texttt{x[]}\ \texttt{:=}\ n\ \{[x = n]\}}\ \textsc{assign-pr}$$

7.2.4 Measure into Bits

The definition of the syntactic sugar to measure into a bit is as such:

$$b\ \texttt{:= measure}\ q \equiv \texttt{measure}\ q\ \texttt{then}\ b\ \texttt{:= 1 else}\ b\ \texttt{:= 0}$$

Since b and q are distinct registers, we have by ASSIGN:

$$\{{}^qE_i[\phi \wedge b = c]\}\ b\ \texttt{:=}\ i\ \{{}^qE_i[\phi \wedge b = i]\}$$

We can also use $q = i$ in the probability predicate, to conclude that $b = q$ in the postcondition:

$$\{{}^qE_i[\phi \wedge b = c]\}\ b\ \texttt{:=}\ i\ \{{}^qE_i[\phi \wedge b = q]\}$$

Then by MEASURE we can combine the branches:

$$\frac{\dfrac{\begin{array}{c}\{{}^qE_0[\phi \wedge b = c]\}\ b\ \texttt{:= 0}\ \{{}^qE_0[\phi \wedge b = q]\}\\ \{{}^qE_1[\phi \wedge b = c]\}\ b\ \texttt{:= 1}\ \{{}^qE_1[\phi \wedge b = q]\}\end{array}}{\{[\phi \wedge b = c]\}\ b\ \texttt{:= measure}\ q\ \{{}^qE_0[\phi \wedge b = q] \oplus {}^qE_1[\phi \wedge b = q]\}}\ \textsc{measure}}{\{[\phi \wedge b = c]\}\ b\ \texttt{:= measure}\ q\ \{[\phi \wedge b = q]\}}\ \textsc{logic}$$

The syntactic sugar for measuring multiple qbits into bits follows:

$$\frac{}{\{[\phi \wedge b = c]\}\ b\texttt{[]}\ \texttt{:= measure}\ q\texttt{[]}\ \{[\phi \wedge b = q]\}}\ \textsc{measure}$$

We may forego recombining ${}^qE_0$ and ${}^qE_1$ if they are not immediately followed by a pr$(\cdot)$ predicate. This is useful in combination with FRAME-ASSIGN-i.

$$\frac{\{{}^qE_0\phi\}\ b\ \texttt{:= 0}\ \{{}^qE_0\psi_0\} \qquad \{{}^qE_1\phi\}\ b\ \texttt{:= 1}\ \{{}^qE_1\psi_1\}}{\{\phi\}\ b\ \texttt{:= measure}\ q\ \{{}^qE_0\psi_0 \oplus {}^qE_1\psi_1\}}\ \textsc{measure}$$

7.2.5 Distribute over E_0 and E_1 with Independent Measurements

We can use independent measurements to reorder measurements, which is useful in nested if statements. If we have some chain of measurements $M = {}^{b_1}E_{i_1} \dots {}^{b_n}E_{i_n}$ where $b_i \neq a$ we can use the following equivalencies:

$$\begin{aligned}
&A \models {}^aE_0M[\rho] \oplus {}^aE_1M[\rho] \\
\text{iff } &A \models M{}^aE_0[\rho] \oplus M{}^aE_1[\rho] \text{ (by Lemma 5)} \\
\text{iff } &A \models M({}^aE_0[\rho] \oplus {}^aE_1[\rho]) \text{ (by Lemma 6)} \\
\text{iff } &A \models M[\rho] \text{ (by Lemma 4)}
\end{aligned}$$

We can then restate IF as:

$$\frac{\begin{array}{c}\{{}^aE_0\phi\}\ P\ \{{}^aE_0M[\rho]\} \\ \{{}^aE_1\phi\}\ Q\ \{{}^aE_1M[\rho]\} \qquad M = {}^{b_1}E_{i_1} \dots {}^{b_n}E_{i_n} \wedge b_i \neq a\end{array}}{\{\phi\}\ \texttt{if}\ a\ \texttt{then}\ P\ \texttt{else}\ Q\ \{M[\rho]\}}\ \text{IF}$$

7.3 Mathematical Lemmas

7.3.1 Zero Diagonal

The interpretation A of a formula ϕ is restricted to be a (partial) density operator, which is a Hermitian positive semi-definite matrix. We want to show that when an element of the diagonal of A is zero ($A_{ii} = 0$), the corresponding row and column are zero as well:

Lemma 7. *For a partial density matrix A: $A_{ii} = 0 \Rightarrow \forall j : A_{ij} = A_{ji} = 0$*

Proof. (based on [1]) The proof is by contradiction. Since A is Hermitian, we have that $A_{ij} = A^\dagger_{ji}$, so $A_{ij} \neq 0$ if and only if $A_{ji} \neq 0$. We continue w.l.o.g. assuming $A_{ij} \neq 0$. Consider this matrix A', a submatrix of A:

$$A' \equiv \begin{bmatrix} 0 & A_{ij} \\ A^\dagger_{ij} & A_{jj} \end{bmatrix}$$

A well-known result is that the determinant of a matrix is the product of its eigenvalues. We can see that $det(A') = 0 \cdot A_{jj} - A^\dagger_{ij} \cdot A_{ij} < 0$, and so one of the eigenvalues of A' is negative. Consequently A' is not positive semi-definite, and we may obtain a vector x' such that $x'^T A' x' < 0$.

Next we extend x' to the size of A, where $x_i = x_1$, $x_j = x_2$ and zero otherwise. Although we started with the assumption that A is positive semi-definite, we have now found a counterexample: $x^T A x = x'^T A' x' < 0$, a contradiction. □

Chapter 8

Proof

This chapter presents the central proof of the thesis: we prove the case study of Shor's factoring algorithm (introduced in Chapter 6) in the logic presented in Chapter 5. To keep the textual size of predicates in the proof trees somewhat manageable, we introduce these shorthands:

- We use the notation $d|N$ to say that d is a *nontrivial* divisor of N: d fits an integer number of times in N $(d \cdot n = N)$, and it is not trivial: $d \neq 1$ and $d \neq N$.
- $N_{\text{std}} \equiv N > 1 \land N$ is not prime $\land N$ is not even $\land N$ is not a prime power $\land P = 0$
- $S_{\text{cases}} \equiv (S = 1 \Rightarrow d = 0) \land (S = 0 \Rightarrow d|N)$
- $Z_{i,j,\cdots} \equiv i = 0 \land j = 0 \land \cdots$

8.1 Proof Tree

The only input to the program is an integer N greater than 1. The language has no notion of dynamic allocation, so we must know in advance how many bits and qbits to allocate. For this we obtain integers q, l such that $q = 2^l$ and $N^2 < q \leq 2N^2$, and integer n such that $N \leq 2^n$. The program we verify is thus parametric over q, l and n, but only for trivial reasons.

We start by declaring all the variables used. d is the divisor of N, P is used to store whether N is a prime power, S is the boolean determining whether we are still searching for a factor, x, a and f_a are the values at the core of Shor's algorithm, b is a as measured, c and r are the fraction expansion of b/q.

$$\frac{\begin{array}{c}
\{[N > 1 \land N \text{ is not prime}]\}\ \texttt{bit d[n]}\ \{[N > 1 \land N \text{ is not prime} \land Z_d]\}\ (\textsc{decld}) \\
\{[N > 1 \land N \text{ is not prime} \land Z_d]\}\ \texttt{bit P[n]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P}]\}\ (\textsc{declp}) \\
\{[N > 1 \land N \text{ is not prime} \land Z_{d,P}]\}\ \texttt{bit S}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S}]\}\ (\textsc{decls}) \\
\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S}]\}\ \texttt{bit x[n]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x}]\}\ (\textsc{declx}) \\
\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x}]\}\ \texttt{qbit a[l]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a}]\}\ (\textsc{decla}) \\
\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a}]\}\ \texttt{qbit fa[n]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,fa}]\}\ (\textsc{declfa}) \\
\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,fa}]\}\ \texttt{bit b[l]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b}]\}\ (\textsc{declb}) \\
\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b}]\}\ \texttt{bit c[l]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c}]\}\ (\textsc{declc}) \\
\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c}]\}\ \texttt{bit r[l]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\ (\textsc{declr})
\end{array}}{\{[N > 1 \land N \text{ is not prime}]\}\ \ldots\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}}\ \textsc{seq} \qquad (\textsc{declare})$$

The declarations can be derived from the NEW-(Q)BIT axioms:

$$\frac{}{\{[N > 1 \land N \text{ is not prime}]\}\ \texttt{bit d[n]}\ \{[N > 1 \land N \text{ is not prime} \land Z_d]\}}\ \textsc{new-bit} \qquad (\textsc{decld})$$

$$\frac{}{\{[N > 1 \land N \text{ is not prime} \land Z_d]\}\ \texttt{bit P[n]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P}]\}}\ \textsc{new-bit} \qquad (\textsc{declp})$$

$$\frac{}{\{[N > 1 \land N \text{ is not prime} \land Z_{d,P}]\}\ \texttt{bit S}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S}]\}}\ \textsc{new-bit} \qquad (\textsc{decls})$$

$$\frac{}{\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S}]\}\ \texttt{bit x[n]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x}]\}}\ \textsc{new-bit} \qquad (\textsc{declx})$$

$$\frac{}{\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x}]\}\ \texttt{qbit a[l]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a}]\}}\ \textsc{new-qbit} \qquad (\textsc{decla})$$

$$\frac{}{\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a}]\}\ \texttt{qbit fa[n]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,fa}]\}}\ \textsc{new-qbit} \qquad (\textsc{declfa})$$

$$\frac{}{\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,fa}]\}\ \texttt{bit b[l]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b}]\}}\ \textsc{new-bit} \qquad (\textsc{declb})$$

$$\frac{}{\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b}]\}\ \texttt{bit c[l]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c}]\}}\ \textsc{new-bit} \qquad (\textsc{declc})$$

$$\frac{}{\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c}]\}\ \texttt{bit r[l]}\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}}\ \textsc{new-bit} \qquad (\textsc{declr})$$

We want to show that d is a non-trivial divisor of N in the postcondition:

$$\dfrac{\begin{array}{c}\{[N > 1 \land N \text{ is not prime}]\}\ \dots\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\ (\textsc{declare})\\ \{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\ \texttt{if N[0] then}\ \dots\ \texttt{else d[] := 2}\ (\text{line 1-24})\ \{[d|N]\}\ (\textsc{ifeven})\end{array}}{\{[N > 1 \land N \text{ is not prime}]\}\ \dots\ (\text{line 1-24})\ \{[d|N]\}}\ \textsc{seq} \qquad (\textsc{all})$$

For Shor's algorithm to work, N must not be even, and not be the power of a single prime. We first establish whether N is even by inspecting its least significant bit:

$$\dfrac{\begin{array}{c}\{{}^{N[0]}E_0[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\ \texttt{d[] := 2}\ (\text{line 2})\ \{{}^{N[0]}E_0[d|N]\}\ (\textsc{even})\\ \{{}^{N[0]}E_1[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\ \dots\ (\text{line 4-23})\ \{{}^{N[0]}E_1[d|N]\}\ (\textsc{noteven})\end{array}}{\{[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\ \texttt{if N[0] then}\ \dots\ \texttt{else d[] := 2}\ (\text{line 1-24})\ \{[d|N]\}}\ \textsc{if} \qquad (\textsc{ifeven})$$

$$\dfrac{\begin{array}{c}\dfrac{}{\{{}^{N[0]}E_0[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\ \texttt{d[] := 2}\ (\text{line 2})\ \{{}^{N[0]}E_0[N > 1 \land N \text{ is not prime} \land Z_{P,S,x,a,f_a,b,c,r} \land d = 2]\}}\ \textsc{assign}\\ {}^{N[0]}E_0[N > 1 \land N \text{ is not prime} \land Z_{P,S,x,a,f_a,b,c,r} \land d = 2]\ \models\ {}^{N[0]}E_0[N > 1 \land N \text{ is not prime} \land d = 2 \land N \text{ is even}]\ \models\ {}^{N[0]}E_0[d|N]\end{array}}{\{{}^{N[0]}E_0[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\ \texttt{d[] := 2}\ (\text{line 2})\ \{{}^{N[0]}E_0[d|N]\}}\ \textsc{logic} \qquad (\textsc{even})$$

We claim without proof that it is possible to determine classically whether or not N is a prime power in the syntax available to us. In the below code the implementation gives us a prime P that divides N if N is a prime power, and 0 otherwise:

$$\dfrac{}{\begin{array}{c}\{{}^{N[0]}E_1[N > 1 \land N \text{ is not prime} \land Z_{d,P,S,x,a,f_a,b,c,r}]\}\\ \dots\ (\text{line 4})\\ \{{}^{N[0]}E_1[N > 1 \land N \text{ is not prime} \land Z_{d,S,x,a,f_a,b,c,r} \land ((P = 0 \land N \text{ is not a prime power}) \lor (P|N \land P \text{ is prime}))]\}\end{array}} \qquad (\textsc{primepower})$$

Next we use $P[0]$ to determine whether we have found such a prime. If we have found it, we can use it as a non-trivial factor of N:

$$\dfrac{\begin{array}{c}\dfrac{}{\begin{array}{c}\{{}^{P[0]}E_1{}^{N[0]}E_1[N > 1 \land N \text{ is not prime} \land Z_{d,S,x,a,f_a,b,c,r} \land ((P = 0 \land N \text{ is not a prime power}) \lor (P|N \land P \text{ is prime}))]\}\\ \texttt{d[] := P[]}\ (\text{line 5})\\ \{{}^{P[0]}E_1{}^{N[0]}E_1[N > 1 \land N \text{ is not prime} \land Z_{S,x,a,f_a,b,c,r} \land d = P \land ((P = 0 \land N \text{ is not a prime power}) \lor (P|N \land P \text{ is prime}))]\}\end{array}}\ \textsc{assign}\\ N > 1 \land N \text{ is not prime} \land d = P \land P \neq 0 \land ((P = 0 \land N \text{ is not a prime power}) \lor (P|N \land P \text{ is prime})) \Rightarrow d|N\end{array}}{\begin{array}{c}\{{}^{P[0]}E_1{}^{N[0]}E_1[N > 1 \land N \text{ is not prime} \land Z_{d,S,x,a,f_a,b,c,r} \land ((P = 0 \land N \text{ is not a prime power}) \lor (P|N \land P \text{ is prime}))]\}\\ \texttt{d[] := P[]}\ (\text{line 5})\\ \{{}^{P[0]}E_1{}^{N[0]}E_1[d|N]\}\end{array}}\ \textsc{logic} \qquad (\textsc{yesprime})$$

$$\frac{\{\dots\}\ \texttt{S := 1; while S do}\ \dots\ \text{(line 7-22)}\ \{\dots\}\ (\textsc{noprime}) \qquad \{\dots\}\ \texttt{d[] := P[]}\ \text{(line 5)}\ \{\dots\}\ (\textsc{yesprime})}{\begin{array}{c}\{{}^{N[0]}E_1[N>1 \wedge N \text{ is not prime} \wedge Z_{d,S,x,a,f_a,b,c,r} \wedge ((P=0 \wedge N \text{ is not a prime power}) \vee (P|N \wedge P \text{ is prime}))]\}\\ \texttt{if P[0] then d[] := P[] else}\ \dots\ \text{(line 4-23)}\\ \{{}^{N[0]}E_1[d|N]\}\end{array}}\ \textsc{if} \qquad (\textsc{ifprimepow})$$

$$\frac{\{\dots\}\ \dots\ \text{(line 4)}\ \{\dots\}\ (\textsc{primepower}) \qquad \{\dots\}\ \texttt{if P[0] then d[] := P[] else}\ \dots\ \text{(line 4-23)}\ \{\dots\}\ (\textsc{ifprimepow})}{\{{}^{N[0]}E_1[N>1 \wedge N \text{ is not prime} \wedge Z_{d,P,S,x,a,f_a,b,c,r}]\}\ \dots\ \text{(line 4-23)}\ \{{}^{N[0]}E_1[d|N]\}}\ \textsc{seq} \qquad (\textsc{noteven})$$

The main loop repeatedly executes the Shor algorithm to produce a non-trivial factor, since the algorithm only produces a factor with high probability. We will declare a boolean `S` (‘searching’) to indicate that d does not yet contain a valid factor.

$$\frac{}{\begin{array}{c}\{{}^{P[0]}E_0{}^{N[0]}E_1[N>1 \wedge N \text{ is not prime} \wedge Z_{d,S,x,a,f_a,b,c,r} \wedge ((P=0 \wedge N \text{ is not a prime power}) \vee (P|N \wedge P \text{ is prime}))]\}\\ \texttt{S := 1}\ \text{(line 7)}\\ \{{}^{P[0]}E_0{}^{N[0]}E_1[N>1 \wedge N \text{ is not prime} \wedge S=1 \wedge Z_{d,x,a,f_a,b,c,r} \wedge ((P=0 \wedge N \text{ is not a prime power}) \vee (P|N \wedge P \text{ is prime}))]\}\end{array}}\ \textsc{assign} \qquad (\textsc{search})$$

While we have not found a factor, we execute the period-finding algorithm. When we have found a factor, it should conform to our final postcondition. The logic gives us the ability to make the pre- and postcondition of the loop body dependent on the iteration number, but we forego this ability and instead use a loop invariant. Our loop therefore looks like this:

$$\frac{\dfrac{\{\dots\}\ \dots\ \text{(line 8-21)}\ \{\dots\}\ (\textsc{loopbodytrue})}{\begin{array}{c}\{{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge Z_{d,x,a,f_a,b,c,r}]\}\\ \texttt{while S do}\ \dots\ \text{(line 7-22)}\\ \{{}^{S}E_0{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge Z_{d,x,a,f_a,b,c,r}]\}\end{array}}\ \textsc{while} \\ {}^{S}E_0{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge Z_{d,x,a,f_a,b,c,r}] \models {}^{S}E_0{}^{P[0]}E_0{}^{N[0]}E_1[S=0 \wedge N_{\text{std}} \wedge S_{\text{cases}} \wedge Z_{d,x,a,f_a,b,c,r}] \models {}^{S}E_0{}^{P[0]}E_0{}^{N[0]}E_1[d|N] \models {}^{P[0]}E_0{}^{N[0]}E_1[d|N]}{\begin{array}{c}\{{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge Z_{d,x,a,f_a,b,c,r}]\}\\ \texttt{while S do}\ \dots\ \text{(line 7-22)}\\ \{{}^{P[0]}E_0{}^{N[0]}E_1[d|N]\}\end{array}}\ \textsc{logic} \qquad (\textsc{loop})$$

$$\cfrac{\begin{array}{c}\{\dots\}\ \texttt{S := 1}\ \text{(line 7)}\ \{\dots\}\ (\text{SEARCH}) \qquad \{\dots\}\ \texttt{while S do ...}\ \text{(line 7-22)}\ \{\dots\}\ (\text{LOOP}) \\ {}^{P[0]}E_0{}^{N[0]}E_1[N > 1 \wedge N \text{ is not prime} \wedge S = 1 \wedge Z_{d,x,a,f_a,b,c,r} \wedge ((P = 0 \wedge N \text{ is not a prime power}) \vee (P|N \wedge P \text{ is prime}))] \\ \models {}^{P[0]}E_0{}^{N[0]}E_1[P_{2,0} = 0 \wedge N_{2,0} = 1 \wedge N > 1 \wedge N \text{ is not prime} \wedge S = 1 \wedge Z_{d,x,a,f_a,b,c,r} \wedge ((P = 0 \wedge N \text{ is not a prime power}) \vee (P|N \wedge P \text{ is prime}))] \\ \models {}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge Z_{d,x,a,f_a,b,c,r}]\end{array}}{\begin{array}{c}\{{}^{P[0]}E_0{}^{N[0]}E_1[N > 1 \wedge N \text{ is not prime} \wedge Z_{d,S,x,a,f_a,b,c,r} \wedge ((P = 0 \wedge N \text{ is not a prime power}) \vee (P|N \wedge P \text{ is prime}))]\} \\ \texttt{S := 1; while S do ...}\ \text{(line 7-22)} \\ \{{}^{P[0]}E_0{}^{N[0]}E_1[d|N]\}\end{array}}\ \text{SEQ}$$

(NOPRIME)

First we must obtain a uniform sample $x \in [2, N)$. The syntactic sugar already contains syntax to sample a fixed number of bits. Since the success of the algorithm is probabilistic to start with, we might as well sample the range $[0, q)$ and retry until a valid sample is found.

$$\cfrac{}{\begin{array}{c}\{{}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge Z_{d,x,a,f_a,b,c,r}]\} \\ \texttt{x[] := rnd [2, N)}\ \text{(line 8)} \\ \{\bigoplus_{x' \in [2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\}\end{array}}\ \text{SAMPLE}$$

(SAMPLEX)

We make each measurement outcome of a equally likely by applying the QFT to it.

$$\cfrac{}{\begin{array}{c}\{\bigoplus_{x' \in [2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\} \\ \texttt{a[] *= QFT}\ \text{(line 11)} \\ \{{}^{a}QFT^{\dagger} \bigoplus_{x' \in [2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\}\end{array}}\ \text{UNITARY}$$

(DISTA)

Next, we apply the quantum operator that computes x^a over the superposition of a:

$$\cfrac{}{\begin{array}{c}\{{}^{a}QFT^{\dagger} \bigoplus_{x' \in [2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\} \\ \texttt{a[], fa[] *= Of}\ \text{(line 12)} \\ \{{}^{a:fa}O_f^{\dagger}{}^{a}QFT^{\dagger} \bigoplus_{x' \in [2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\}\end{array}}\ \text{UNITARY}$$

(APPLYMAGIC)

We measure f_a, the outcome of x^a:

$$\frac{}{\begin{array}{c}\{{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{x'\in[2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\} \\ \texttt{measure fa[]}\ \text{(line 13)} \\ \{\bigoplus\limits_{a'\in[0,N)} {}^{fa}E_{a'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{x'\in[2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\}\end{array}}\ \textsc{measure} \qquad (\textsc{measurefa})$$

Finally, we apply the QFT to a again:

$$\frac{}{\begin{array}{c}\{\bigoplus\limits_{fa'\in[0,N)} {}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{x'\in[2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\} \\ \texttt{a[] *= QFT}\ \text{(line 14)} \\ \{{}^{a}QFT^{\dagger} \bigoplus\limits_{fa'\in[0,N)} {}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{x'\in[2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\}\end{array}}\ \textsc{unitary} \qquad (\textsc{qfta})$$

When we now measure a, we are likely to get a sample that will produce a factor for us:

$$\frac{}{\begin{array}{c}\{{}^{a}QFT^{\dagger} \bigoplus\limits_{fa'\in[0,N)} {}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{x'\in[2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge Z_{d,a,f_a,b,c,r}]\} \\ \texttt{b[] := measure a[]}\ \text{(line 15)} \\ \{\bigoplus\limits_{a'\in[0,q)} {}^{a}E_{a'}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{fa'\in[0,N)} {}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{x'\in[2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge b = a' \wedge Z_{d,a,f_a,c,r}]\}\end{array}}\ \textsc{measure} \qquad (\textsc{measurea})$$

Using our obtained sample b we perform continued fraction expansion on the fraction b/q. This is a classical problem, so we will not implement it here. Instead we will assume we obtain a fraction c/r, where we are interested in r: the likely candidate for the period of the modular exponentiation of x.

$$\frac{}{\begin{array}{c}\{\bigoplus\limits_{a'\in[0,q)} {}^{a}E_{a'}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{fa'\in[0,N)} {}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{x'\in[2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge b = a' \wedge Z_{d,a,f_a,c,r}]\} \\ \texttt{CFE}\ \text{(line 16)} \\ \{\bigoplus\limits_{a'\in[0,q)} {}^{a}E_{a'}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{fa'\in[0,N)} {}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus\limits_{x'\in[2,N)} {}^{S}E_1{}^{P[0]}E_0{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge b = a' \wedge |b/q - c/r| \le 1/2q \wedge Z_{d,a,f_a}]\}\end{array}}\ \textsc{assumed} \qquad (\textsc{cfe})$$

Finally, it is likely that r yields a non-trivial factor of N: it is likely that either $gcd(x^{r/2}-1,N)$ or $gcd(x^{r/2}+1,N)$ is a factor of N greater than 1. Determining the greatest common divisor is once again a classical process, so we omit it here:

$$\dfrac{}{\begin{array}{c}\{\bigoplus_{a'\in[0,q)} {}^{a}E_{a'}\,{}^{a}QFT^{\dagger} \bigoplus_{fa'\in[0,N)} {}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus_{x'\in[2,N)} {}^{S}E_1\,{}^{P[0]}E_0\,{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge b = a' \wedge |b/q - c/r| \leq 1/2q \wedge Z_{d,a,f_a}]\} \\ \texttt{GCD}\text{ (line 17-21)} \\ \{\bigoplus_{a'\in[0,q)} {}^{a}E_{a'}\,{}^{a}QFT^{\dagger} \bigoplus_{fa'\in[0,N)} {}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger} \bigoplus_{x'\in[2,N)} {}^{S}E_1\,{}^{P[0]}E_0\,{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge b = a' \wedge |b/q - c/r| \leq 1/2q \wedge Z_{d,a,f_a}]\}\end{array}} \text{ ASSUMED}$$

(GCDRESULT)

The last step is to simply combine the steps of the core algorithm.

$$\dfrac{\begin{array}{c}\{\ldots\}\ \texttt{x[] := rnd [2, N)}\text{ (line 8) }\{\ldots\}\text{ (SAMPLEX)} \qquad \{\ldots\}\ \texttt{a[] *= QFT}\text{ (line 11) }\{\ldots\}\text{ (DISTA)} \\ \{\ldots\}\ \texttt{a[], fa[] *= Of}\text{ (line 12) }\{\ldots\}\text{ (APPLYMAGIC)} \qquad \{\ldots\}\ \texttt{measure fa[]}\text{ (line 13) }\{\ldots\}\text{ (MEASUREFA)} \qquad \{\ldots\}\ \texttt{a[] *= QFT}\text{ (line 14) }\{\ldots\}\text{ (QFTA)} \\ \{\ldots\}\ \texttt{b[] := measure a[]}\text{ (line 15) }\{\ldots\}\text{ (MEASUREA)} \qquad \{\ldots\}\ \texttt{CFE}\text{ (line 16) }\{\ldots\}\text{ (CFE)} \qquad \{\ldots\}\ \texttt{GCD}\text{ (line 17-21) }\{\ldots\}\text{ (GCDRESULT)}\end{array}}{\begin{array}{c}\{{}^{S}E_1\,{}^{P[0]}E_0\,{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}} \wedge Z_{d,x,a,f_a,b,c,r}]\} \\ \ldots\text{ (line 8-21)} \\ \{{}^{P[0]}E_0\,{}^{N[0]}E_1[N_{\text{std}} \wedge S_{\text{cases}}]\}\end{array}} \text{ SEQ}$$

(LOOPBODYTRUE)

8.2 Comparison to Theory

The result of continued fraction expansion (CFE) and obtaining an actual factor from the GCD (GCDRESULT) is assumed. We should therefore make sure that we have actually established sufficient conditions for Shor's period-finding algorithm to be likely to have worked. To show this we compare the probability of a measurement outcome b in the precondition of CFE to the corresponding state in Shor's period-finding algorithm. Before CFE our program is in some state A satisfying:

$$A \models \bigoplus_{a' \in [0,q)} {}^{a}E_{a'} {}^{a}QFT^{\dagger} \bigoplus_{fa' \in [0,N)} {}^{fa}E_{fa'} {}^{a:fa}O_f^{\dagger} {}^{a}QFT^{\dagger} \bigoplus_{x' \in [2,N)} {}^{S}E_1 {}^{P[0]}E_0 {}^{N[0]}E_1 [\ldots]$$

Interpreting the formula with A, and considering that matrix multiplication (transformations) distribute over addition (probabilistic sum $\oplus$), we can ascertain that this is equivalent to:

$$\begin{aligned} A &= \sum_{a' \in [0,q)} \sum_{f'_a \in [0,N)} \sum_{x' \in [2,N)} A(a', f'_a, x') \\ A(a', f'_a, x') &\models {}^{a}E_{a'} {}^{a}QFT^{\dagger} {}^{fa}E_{fa'} {}^{a:fa}O_f^{\dagger} {}^{a}QFT^{\dagger} {}^{S}E_1 {}^{P[0]}E_0 {}^{N[0]}E_1 \\ &\quad [N_{\text{std}} \wedge S_{\text{cases}} \wedge x = x' \wedge b = a' \wedge Z_{d,a,f_a,c,r}] \end{aligned}$$

We can establish that $A(a', f'_a, x')$ has non-zero entries only where $b = a'$. The total probability of $\sum_{f'_a} \sum_{x'} A(a', f'_a, x')$ then represents the probability of measurement outcome b. To show the equivalence with the state in the theoretical algorithm, we use Dirac notation. This notation corresponds with program states A when we choose the standard basis e_0, e_1. We start with a system that satisfies the probability predicate, and apply the transformations outward from there. Unfolding N_{std} and S_{cases} we can find that $a = 0$, $f_a = 0$, $d = 0$, $P = 0$, $N[0] = 0$ and $S = 1$. Consequently, diagonal entries in $A(a', f'_a, x')$ that do not correspond to these conditions are zero, as are the corresponding rows and columns by Lemma 7. The system is thus in the form:

$$c|0\rangle_a\langle 0| \otimes |0\rangle_{f_a}\langle 0| \otimes |0\rangle_d\langle 0| \otimes |0\rangle_P\langle 0| \otimes |0\rangle_{N[0]}\langle 0| \otimes |1\rangle_S\langle 1| \otimes T$$

The transformations in the formula do not affect or depend on T, so we do not consider it here. We are interested in the result:

$$\begin{aligned} & {}^{a}E_{a'} {}^{a}QFT^{\dagger} {}^{fa}E_{fa'} {}^{a:fa}O_f^{\dagger} {}^{a}QFT^{\dagger} {}^{S}E_1 {}^{P[0]}E_0 {}^{N[0]}E_1 \\ & c|0\rangle_a\langle 0| \otimes |0\rangle_{f_a}\langle 0| \otimes |0\rangle_d\langle 0| \otimes |0\rangle_P\langle 0| \otimes |0\rangle_{N[0]}\langle 0| \otimes |1\rangle_S\langle 1| \\ & {}^{a}E^{\dagger}_{a'} {}^{a}QFT {}^{fa}E^{\dagger}_{fa'} {}^{a:fa}O_f {}^{a}QFT {}^{S}E^{\dagger}_1 {}^{P[0]}E^{\dagger}_0 {}^{N[0]}E^{\dagger}_1 \\ =& cvv^{\dagger} \end{aligned}$$

For ease of computation we define v as follows:

$$\begin{aligned}
v =& {}^{a}E_{a'}\,{}^{a}QFT^{\dagger}\,{}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger}\,{}^{S}E_1\,{}^{P[0]}E_0\,{}^{N[0]}E_1|0\rangle_a|0\rangle_{f_a}|0\rangle_d|0\rangle_P|0\rangle_{N[0]}|1\rangle_S \\
=& {}^{a}E_{a'}\,{}^{a}QFT^{\dagger}\,{}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\,{}^{a}QFT^{\dagger}|0\rangle_a|0\rangle_{f_a}|0\rangle_d|0\rangle_P|0\rangle_{N[0]}|1\rangle_S \\
=& {}^{a}E_{a'}\,{}^{a}QFT^{\dagger}\,{}^{fa}E_{fa'}\,{}^{a:fa}O_f^{\dagger}\frac{1}{\sqrt{q}}\sum_{a=0}^{q-1}|a\rangle_a|0\rangle_{f_a}|0\rangle_d|0\rangle_P|0\rangle_{N[0]}|1\rangle_S \\
=& {}^{a}E_{a'}\,{}^{a}QFT^{\dagger}\,{}^{fa}E_{fa'}\frac{1}{\sqrt{q}}\sum_{a=0}^{q-1}|a\rangle_a|f(a)\rangle_{f_a}|0\rangle_d|0\rangle_P|0\rangle_{N[0]}|1\rangle_S \\
=& {}^{a}E_{a'}\,{}^{a}QFT^{\dagger}\frac{1}{\sqrt{m}}\sum_{j=0}^{m-1}|jr+s\rangle_a|f_a'\rangle_{f_a}|0\rangle_d|0\rangle_P|0\rangle_{N[0]}|1\rangle_S \\
=& {}^{a}E_{a'}\frac{1}{\sqrt{mq}}\sum_{j=0}^{m-1}\sum_{b=0}^{q-1}(e^{2\pi i\frac{(jr+s)b}{q}}|b\rangle_a)|f_a'\rangle_{f_a}|0\rangle_d|0\rangle_P|0\rangle_{N[0]}|1\rangle_S \\
=& \frac{1}{\sqrt{mq}}\left(\sum_{j=0}^{m-1}e^{2\pi i\frac{(jr+s)a'}{q}}\right)|a'\rangle_a|f_a'\rangle_{f_a}|0\rangle_d|0\rangle_P|0\rangle_{N[0]}|1\rangle_S
\end{aligned}$$

Thus, given a value of x, the probability of the system matches that of the theory in Equation (6.1). Note that we are being imprecise about x here: it should be uniformly distributed over $[2, N)$. With the approach we have chosen ('valid' probabilistic predicates) this is hard to specify, so we have chosen not to solve this.

Chapter 9

Conclusion

In this work we have performed a case study that illustrates the usability of a logic that reasons about the correctness of quantum programs. This chapter discusses the findings we learned from this effort, and what is left for future work. Finally we conclude by summarizing the thesis.

9.1 Findings

The case study of Shor's factoring algorithm shows that it is in fact possible to prove a non-trivial example in the logic. This did however require a fair number of lemmas. On balance the logic is certainly usable, but it could be improved by rules that improve the completeness of the logic.

The lemmas do hint at some possible avenues for improvement. The formulae in the logic cannot directly state properties about the state of the system (e.g. "this register is 1"). This is however possible in probability predicates, where we have to be precise about the probability that a statement about the system is true. A trick we then use is that of valid probability predicates: a statement of the form $\mathrm{pr}(\rho) = \mathrm{pr}(\mathsf{true})$. If we specify everything this way we do lose total correctness, but it is much more natural to specify properties about the system in this way.

Other lemmas focus on more fundamental additions. For example, we proved properties like transformations being distributive over probabilistic addition ($\oplus$), and reordering of independent measurements. These are not new insights about quantum systems, but could be good additions to the logic in some form.

Finally we give some rules for the verification of syntactic sugar introduced in the logic. While these are fairly obvious, it is still good to make them precise. In addition, it would be good to make precise the equivalence between multi-qbit registers and bounded integers.

9.2 Future Work

9.2.1 Formalization of Lemmas and Proof

Some of the lemmas, and the proof itself are quite involved. The presented proofs in the thesis do not require particularly deep mathematical insight: the difficulty is rather in finding lemmas (or from the perspective of the proof system: axioms and proof rules) that assist with the proof of an algorithm. In fact, proofs such as the one in Section 7.1.3 are almost mechanical. For that reason, and to further increase the confidence in the correctness of these proofs, it might be a good idea to formalize them in a theorem prover.

9.2.2 Non-Probabilistic Classical Computing

Since quantum computing is fundamentally probabilistic, the quantum programming language is probabilistic as well. This however drags the classical part of the language into the probabilistic domain as well. Writing probabilistic specifications for classical parts that are not probabilistic is cumbersome. To increase usability it would be better if there was some notion of embedding a probabilistic program into a deterministic program.

9.2.3 Complete Quantum Logic

This thesis proposes some lemmas that may contribute towards making a more complete axiomatization of the quantum programming language, but more work is needed to make it truly complete.

9.2.4 Automated Reasoning about Quantum Programs

Some predicates in the central proof are simultaneously quite long, and entirely mechanical. As an example, the rule for unitary operations is complete and requires no choice, and thus is a good candidate for automation. Following the tradition of tools based on Hoare logic, we could pair the programming language with a specification language and automate as much of the proof as possible.

9.3 Summary

In Chapter 2 we gave an introduction to formal program verification. An example logic was given for a small toy language called WHILE.

Chapter 3 introduced the important concepts of quantum computing. For the characterization we use the five criteria as laid out in [6]. Furthermore we explain a foundation for the math of quantum systems.

Having explained program verification and quantum computing Chapter 4 gives an intuition for the structure of several proposed logics designed for verification of quantum algorithms. In the next chapter (Chapter 5) one of these was fully explained, as this is the logic that was used for the case study.

In Chapter 6 we explained the algorithm that was used as a case study: Shor's factoring algorithm. We gave a sketch of the proof why this algorithm works, and lay out a pseudocode implementation of the algorithm, later to be translated to the toy language of the logic from Chapter 5.

To complete the proof a fair number of additional lemmas were needed. This is expected, as the logic introduced in [9] makes no claim to be complete, and remarks in its section on future work that more work is needed to make a complete axiomatization. The lemmas required for our proof are explained in Chapter 7.

Finally, the case study is formally verified in Chapter 8. Although the program indeed provides the desirable result (d is a non-trivial divisor of N), to further establish the correspondence to the algorithm the state of the program in the core part of the factor-finding loop is compared to the theoretical state as explained in Chapter 6.

Bibliography

[1] EuYu (https://math.stackexchange.com/users/9246/euyu). *Zeros diagonal element of a semidefinite matrix leads to zeros row/column. Why?* Feb. 10, 2014. URL: https://math.stackexchange.com/q/671156 (visited on 05/24/2021).

[2] Nick Benton. "Simple relational correctness proofs for static analyses and program transformations". In: *ACM SIGPLAN Notices* 39.1 (2004), pp. 14–25.

[3] Arne Johan Brentjes. "Multi-dimensional continued fraction algorithms". In: *MC Tracts* (1981). URL: https://ir.cwi.nl/pub/13005.

[4] JI Den Hartog and Erik P de Vink. "Verifying probabilistic programs using a Hoare like logic". In: *International journal of foundations of computer science* 13.03 (2002), pp. 315–340.

[5] Paul Adrien Maurice Dirac. "A new notation for quantum mechanics". In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 35. 3. Cambridge University Press. 1939, pp. 416–418.

[6] David P DiVincenzo. "The physical implementation of quantum computation". In: *Fortschritte der Physik: Progress of Physics* 48.9-11 (2000), pp. 771–783.

[7] Charles Antony Richard Hoare. "An axiomatic basis for computer programming". In: *Communications of the ACM* 12.10 (1969), pp. 576–580.

[8] Stephen Joran. *Quantum Algorithm Zoo*. Feb. 1, 2021. URL: https://quantumalgorithmzoo.org/ (visited on 05/11/2021).

[9] Yoshihiko Kakutani. "A logic for formal verification of quantum programs". In: *Annual Asian Computing Science Conference*. Springer. 2009, pp. 79–93.

[10] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[11] Dominique Unruh. "Quantum relational Hoare logic". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–31.

[12] Ronald de Wolf. *Quantum Computing: Lecture Notes*. Jan. 20, 2021. URL: https://homepages.cwi.nl/~rdewolf/qcnotes.pdf (visited on 05/03/2021).

[13] Mingsheng Ying. “Floyd–hoare logic for quantum programs”. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 33.6 (2012), pp. 1–49.