Verifying quantum phase estimation using an expressive theorem-proving assistant

Wayne M. Witzel[1] Warren D. Craft[1,2] Robert Carr[2] and Deepak Kapur[2]
[1]*Center for Computing Research, Quantum Computer Science, Sandia National Laboratories, Albuquerque, New Mexico 87123, USA*
[2]*Department of Computer Science, The University of New Mexico, Albuquerque, New Mexico 87131, USA*

(Received 7 April 2023; revised 28 June 2023; accepted 30 June 2023; published 20 November 2023)

The general-purpose interactive theorem-proving assistant called PROVE-IT was used to verify the quantum phase estimation (QPE) algorithm, specifically claims about its outcome probabilities. PROVE-IT is unique in its ability to express sophisticated mathematical statements, including statements about quantum circuits, integrated firmly within its formal theorem-proving framework. We demonstrate our ability to follow a textbook proof to produce a formally certified proof, highlighting useful automation features to fill in obvious steps and make formal proving nearly as straightforward as informal theorem proving. Finally, we make comparisons with formal theorem proving in other systems where similar claims about QPE have been proven.

DOI: 10.1103/PhysRevA.108.052609

I. INTRODUCTION AND MOTIVATION

Quantum devices in a variety of technologies have been developing rapidly. Several small quantum processors and test beds are now cloud accessible using quantum bits encoded in superconductors [1–4], ions [5,6], neutral atoms [7], photons [8,9], and electron spins [10]. Although current devices do not yet have the size and fidelity for practical applications to definitively compete against classical computers, quantum computers are now a reality and their eventual utility seems inevitable. As a result, there has been widespread and accelerating interest in quantum computers and quantum computing not only in academia and research laboratories, but also in government and industry (as evidenced by the efforts of leaders in the computing industry and computational resource use or allocation such as IBM, Amazon, Intel, Google, and Facebook). Such interest is driven largely by the fact that quantum computers are theorized to have an advantage over classical computers for many computationally difficult problems where, remarkably, the computational advantage grows exponentially with the size of the problem. One well-known example of such an exponential advantage over the best-known classical algorithm is Shor's factoring algorithm [11]. In Shor's approach, factoring is reduced to order finding (finding the smallest positive r for which $x^r = 1 \bmod N$ given x and N) and order finding is reduced to quantum phase estimation (QPE). It is believed that no efficient classical algorithms for factoring and period finding exist. The hidden Abelian subgroup [12] problem is a generalization that may also be solved efficiently on a quantum computer using QPE. Furthermore, QPE has important applications for solving systems of linear equations [13] and performing quantum simulations for chemistry applications [14].

Just as in the case of classical-computing algorithms and software implementing them for classical computers, the reliability and correctness of quantum algorithms and their implementations are vital. Quantum circuits can be complex and often counterintuitive. Analogous to classical software, especially during a time when devices may not exist to test quantum algorithms on a convincing scale, it is critically important to seek rigorous forms of confirmation of both the fundamental concepts of an algorithm and its implementation in a specific device of interest. The eventual physical instantiation of quantum computations and quantum circuits represents substantial physical, financial, and temporal resources. As devices become sufficiently advanced for testing purposes, it is furthermore valuable to ensure algorithms are valid and implemented correctly so there is no confusion between faulty programming versus faulty equipment. This is especially important given the inherent challenge of debugging the implementation of a quantum algorithm running on a quantum computer which can have exponentially many states in the number of qubits, with destructive and probabilistic aspects to any related quantum measurement [15]. It can be economically beneficial to be able to explore quantum circuits and the computational implications of quantum circuits with relative ease using tools based on formal analysis.

At a time when there is a great deal of interest in the potential advantage that may be offered by quantum computers of the future, quantum algorithm verification is very important. This paper applies some unique and promising verification techniques to one quantum algorithm as a demonstration, but there is an entire "zoo" [16] of algorithms with potential quantum advantages that would benefit from formal verification. Our approach to formal verification not only would bring confidence, but could also accelerate the development of new quantum algorithms by preventing costly conceptual mistakes that lead to dead ends and by enabling experts to boldly explore possibilities without fear of getting things wrong. As we hope to demonstrate, PROVE-IT provides a flexibility, expressiveness, and organizational structure to help empower researchers to naturally formulate and test their ideas.

A. Verifying quantum algorithms and quantum circuits

Using formal methods to verify quantum algorithms is not new. A number of languages, frameworks, and environments have been proposed to support the formal analysis and verification of quantum algorithms and quantum circuits. Some examples include VOQC, a fully verified optimizer for quantum circuits, written using the COQ proof assistant, and a low-level language called SQIR (a simple quantum intermediate representation) for expressing quantum circuits as programs [17,18]; QWIRE [19–21], a quantum circuit language embedded in the COQ proof assistant [22,23], allowing users to write quantum circuits using high-level abstractions and to prove properties of those circuits using COQ's theorem-proving features; QBRICKS [24,25], an environment for automated formal verification of quantum programs, taking advantage of the characterization of quantum circuits as parametrized path sums; and quantum Hoare logic [26–30], an extension of Hoare logic for reasoning about quantum programs.

B. The PROVE-IT theorem-proving assistant

This paper reports on a verification of the quantum phase estimation algorithm (see [31], pp. 221–226) using a theorem-proving assistant called PROVE-IT [32,33]. The verification effort has been developed simultaneously with the evolution of PROVE-IT, with the two intertwined activities contributing to each other. The PROVE-IT approach is unique in its ability to construct a formal proof from a user's proof sketch using standard mathematical language with precise semantics based on set theory, thus providing considerable flexibility to a user. We demonstrate this ability in this paper by performing a nearly direct translation of a textbook proof [31] of the QPE algorithm into PROVE-IT. A main objective has been to allow verification in a reasonably accessible way: allowing work by subject matter experts (physicists, engineers, mathematicians, etc.), without extensive background in formal methods, to explore and generate proofs. The automation capabilities of the PROVE-IT system allow the user to skip obvious steps, with PROVE-IT filling in the gaps.

PROVE-IT is an open-source PYTHON-based interactive theorem-proving assistant with a JUPYTER-notebook-based interface. PROVE-IT uses LATEX formatting for output of mathematical expressions, including the mathematical expressions appearing in any eventual formal proof. User-supplied interactive proof steps within a JUPYTER notebook, similar to steps found in a textbook proof or an informal narrative proof, guide PROVE-IT to produce a detailed formal (but entirely human-readable) proof. A small example of this process appears in Figs. 1 and 2, showing the interactive steps (Fig. 1) and eventual PROVE-IT-generated formal proof (Fig. 2) of a closure theorem for modular addition. This proof process is discussed in more detail in Sec. IV.

HTML representations of PROVE-IT system components and related documentation, including all of the code, definitions, axioms, lemmas, and theorems discussed here in relation to the QPE verification efforts, are available for browsing at the PROVE-IT public website [33]. The system is available for download and user contributions at the project's GitHub site [34]. The system is organized hierarchically into theory packages. Each theory package contains axioms and theorems (or lemmas, which are treated the same as theorems). If a theorem has been proven, its name will hyperlink to a proof notebook containing a sequence of commands and a detailed formal proof after the `%qed` command. This proof notebook will also have header information in the top few cells that includes a link to the theorem's dependencies, which lists all axioms and conjectures required by the formal proof, as well as a list of the theorems that directly rely on that particular theorem in their proofs.

C. Organization of the paper

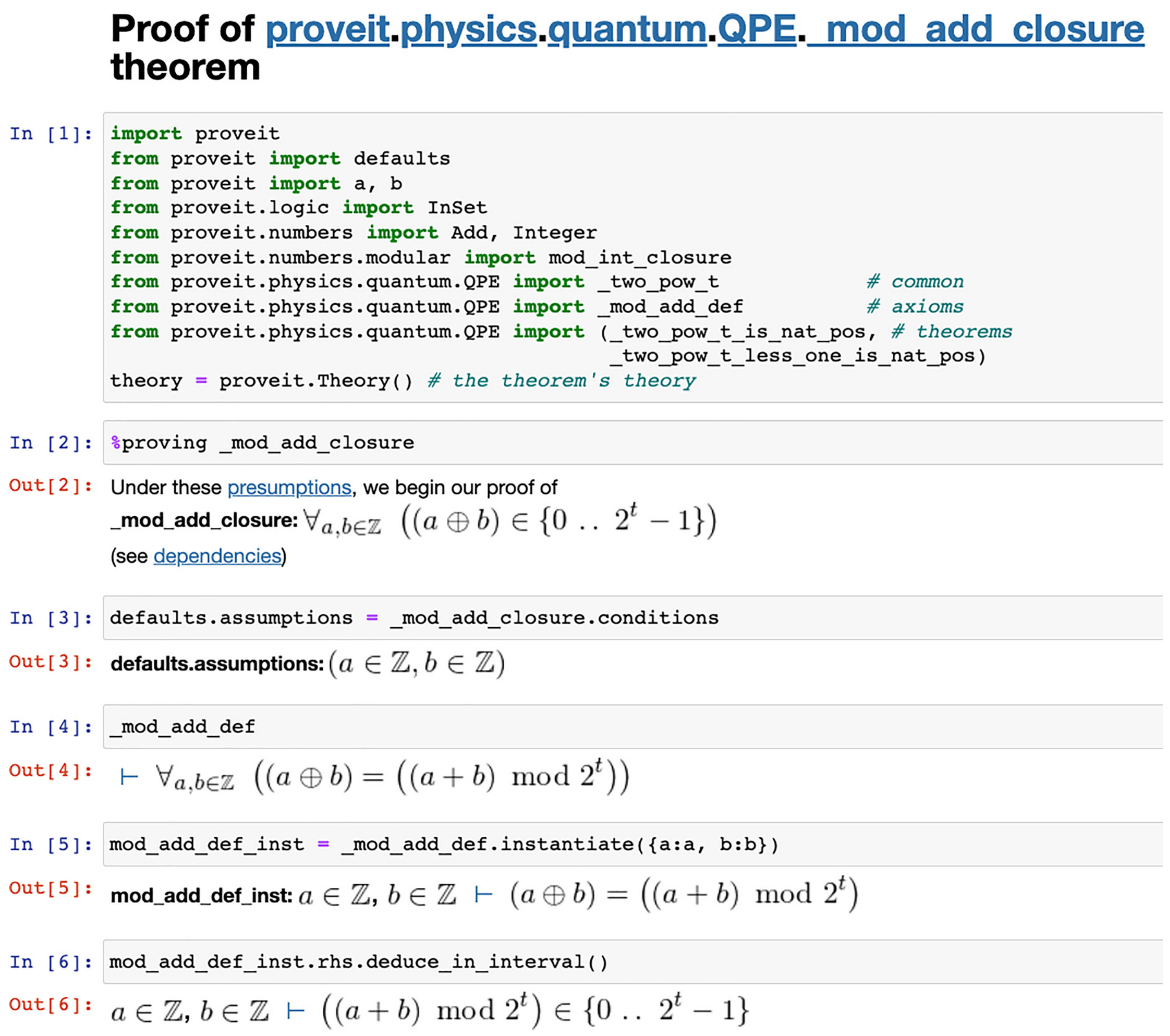
The next three sections of this paper will address the following questions: What, essentially, did we prove (Sec. II)? What, precisely, did we prove (Sec. III)? How did we construct our proofs (Sec. IV)? These topics are brought together in a demonstration of one specific proof within the QPE package in Sec. V. In Sec. VI we make comparisons with other theorem-proving approaches that have been applied to the specific QPE problem. A discussion and conclusions are provided in Sec. VII.

Section II provides an overview of PROVE-IT's QPE theory package, outlining the step-by-step development of the eventual proof for a set of three useful probability bounds for the output from an ideal (error-free) realization of the quantum algorithm: (1) a probability of 1 to obtain the phase when it can represented exactly, (2) a probability greater than $4/\pi^2$ for the result to be the closest solution that can be represented, and (3) the required number of qubits to obtain any desired precision and minimum probability. We progress, in this section, from initial assumptions and axiomatic definitions to final theorems. Section II describes the essence of the QPE package proofs, listing the steps as named statements within a narrative that describes the reasoning to form our conclusions.

Section III drills down to discuss more precisely just what we have proven. We explain how PROVE-IT traces the proven lemmas or theorems back to the axioms and conjectures[1] on which they depend and how PROVE-IT avoids circular dependencies. We also provide specific information about the lemmas or theorems that are proven in our QPE theory package and correspond to the named statements of Sec. II. We list other theory packages that our proofs rely upon and the precise number of axioms and conjectures that were used. We expect that many of the statements that we call axioms at this time will be transformed into conservative definitions [35], with proven unique existence, to reduce the dependencies to a small number of true axioms (e.g., the Zermelo-Fraenkel set theory axioms and postulates of quantum mechanics). At this time, however, we can only claim our proofs to be true conditional upon our current axioms and conjectures.

Section IV discusses in more detail just *how* we proved the theorems interactively, discussing convenient methods and automation features that allow a user to skip obvious steps. We provide a number of examples of using derivation commands to push our QPE-specific proofs along. For each of the

[1]A conjecture in PROVE-IT is any believed statement that is not yet proven in the system; once proven, such a statement becomes a theorem in PROVE-IT. For example, in our QPE proofs we rely on properties of the inverse Fourier transform that are well known but do not have proofs in our system at this time.



Proof of proveit.physics.quantum.QPE._mod_add_closure theorem

In [1]:
```
import proveit
from proveit import defaults
from proveit import a, b
from proveit.logic import InSet
from proveit.numbers import Add, Integer
from proveit.numbers.modular import mod_int_closure
from proveit.physics.quantum.QPE import _two_pow_t          # common
from proveit.physics.quantum.QPE import _mod_add_def        # axioms
from proveit.physics.quantum.QPE import (_two_pow_t_is_nat_pos, # theorems
                                          _two_pow_t_less_one_is_nat_pos)
theory = proveit.Theory() # the theorem's theory
```

In [2]:
```
%proving _mod_add_closure
```

Out[2]: Under these presumptions, we begin our proof of
_mod_add_closure: $\forall_{a,b \in \mathbb{Z}}\ \left((a \oplus b) \in \{0\ ..\ 2^t - 1\}\right)$
(see dependencies)

In [3]:
```
defaults.assumptions = _mod_add_closure.conditions
```

Out[3]: **defaults.assumptions:** $(a \in \mathbb{Z}, b \in \mathbb{Z})$

In [4]:
```
_mod_add_def
```

Out[4]: $\vdash\ \forall_{a,b \in \mathbb{Z}}\ \left((a \oplus b) = \left((a + b) \mod 2^t\right)\right)$

In [5]:
```
mod_add_def_inst = _mod_add_def.instantiate({a:a, b:b})
```

Out[5]: **mod_add_def_inst:** $a \in \mathbb{Z}, b \in \mathbb{Z}\ \vdash\ (a \oplus b) = \left((a + b) \mod 2^t\right)$

In [6]:
```
mod_add_def_inst.rhs.deduce_in_interval()
```

Out[6]: $a \in \mathbb{Z}, b \in \mathbb{Z}\ \vdash\ \left((a + b) \mod 2^t\right) \in \{0\ ..\ 2^t - 1\}$

FIG. 1. Excerpt from the proof notebook for the `_mod_add_closure` (local QPE) theorem, showing the interactive steps (`In`/`Out` cells 1–6) that lead to the PROVE-IT-generated formal proof shown in Fig. 2.

QPE-specific lemmas or theorems (corresponding to named statements of Sec. II) we list the number of derivation commands used in its interactive proof as well as the number of steps in its formal proof.

We focus on one specific supporting, QPE-specific proof in Sec. V to demonstrate PROVE-IT's features in one concrete example. Then in Sec. VI we make specific comparisons with other theorem-proving systems that have been used to prove two out of the three high-level QPE-specific theorems that we accomplished. We close with a more general discussion in Sec. VII.

II. WHAT WE ESSENTIALLY PROVED

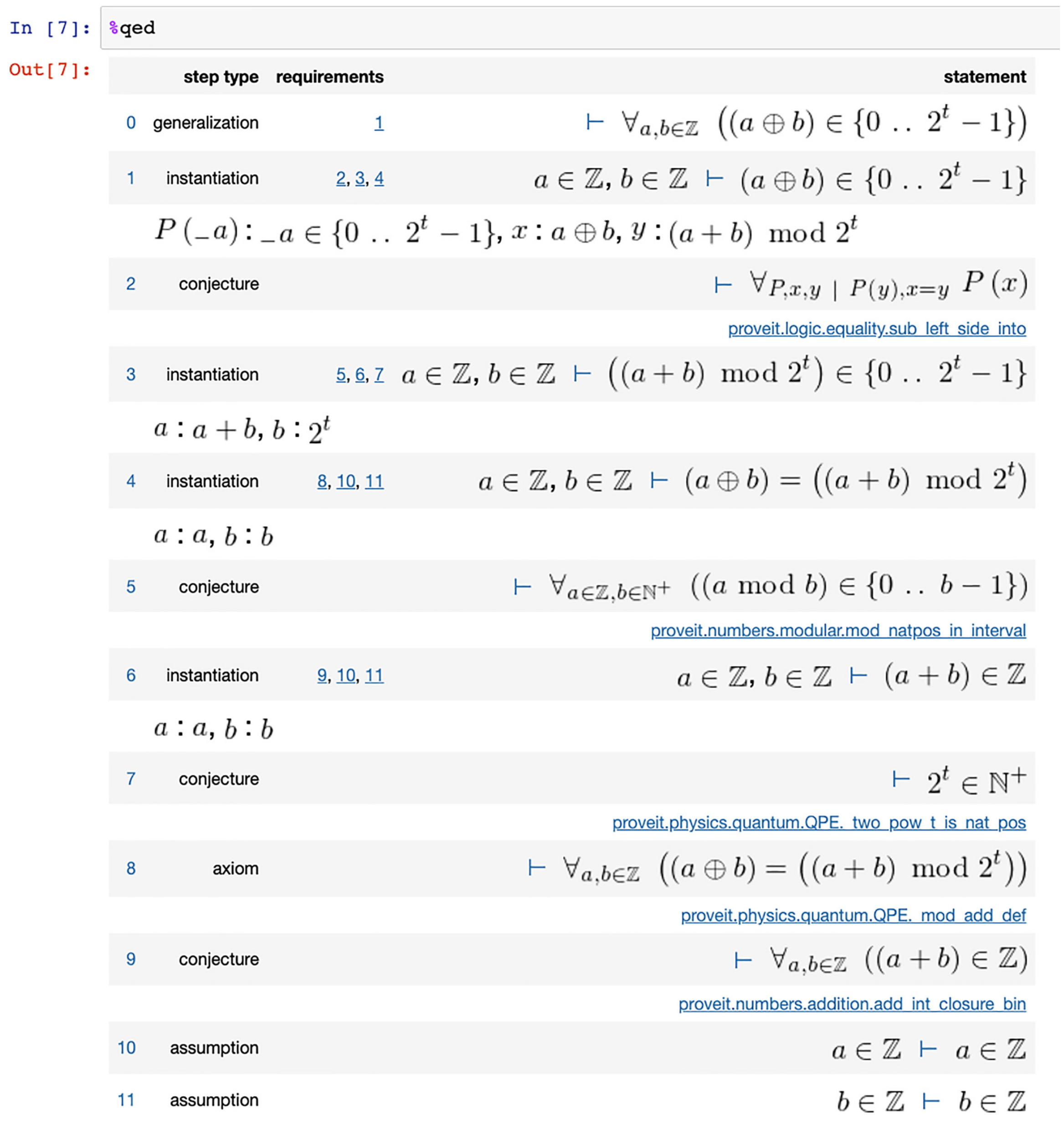
This section provides an overview of the QPE theory package in PROVE-IT and helps the reader understand and navigate the flow of named statements (assumptions, definitions, lemmas, and theorems) within this theory package (as axioms and theorems) on PROVE-IT's public website [33]. We currently treat definitions as axioms in our system which are asserted without proof, but we eventually intend to make a separate category for conservative definitions (see [35], pp. 57–61) which require a proof of unique existence; this change will reduce the number of proof dependencies.

PROVE-IT's QPE theory package can be seen as a formalization of Nielsen and Chuang's [31] standard textbook development and eventual proof of three main claims about the QPE outcome probabilities (also see [36]). We therefore illustrate how PROVE-IT provides an accessible way to construct a formal proof in much the same way one goes about constructing an informal proof. The informal-to-formal transformation is achieved through user-interactive steps that will be discussed in Sec. IV.

In QPE we have a unitary matrix U such that

$$U|u\rangle = e^{2\pi i\varphi}|u\rangle, \tag{1}$$

i.e., U has eigenvector $|u\rangle$ and associated eigenvalue $e^{2\pi i\varphi}$, and we want to estimate the value of the phase $\varphi \in [0, 1)$. The QPE algorithm is shown in quantum circuit form in Fig. 3. The first register has t qubits, with an initial input $|0\rangle_t$; the second register has as many qubits as required to hold the



```
In [7]: %qed
Out[7]:
```

	step type	requirements	statement
0	generalization	1	$\vdash \forall_{a,b \in \mathbb{Z}} \left((a \oplus b) \in \{0 \,..\, 2^t - 1\}\right)$
1	instantiation	2, 3, 4	$a \in \mathbb{Z}, b \in \mathbb{Z} \vdash (a \oplus b) \in \{0 \,..\, 2^t - 1\}$
			$P(\_a) : \_a \in \{0 \,..\, 2^t - 1\}, x : a \oplus b, y : (a+b) \bmod 2^t$
2	conjecture		$\vdash \forall_{P,x,y \,\mid\, P(y), x=y} P(x)$
			proveit.logic.equality.sub_left_side_into
3	instantiation	5, 6, 7	$a \in \mathbb{Z}, b \in \mathbb{Z} \vdash \left((a+b) \bmod 2^t\right) \in \{0 \,..\, 2^t - 1\}$
			$a : a+b, b : 2^t$
4	instantiation	8, 10, 11	$a \in \mathbb{Z}, b \in \mathbb{Z} \vdash (a \oplus b) = \left((a+b) \bmod 2^t\right)$
			$a : a, b : b$
5	conjecture		$\vdash \forall_{a \in \mathbb{Z}, b \in \mathbb{N}^+} \left((a \bmod b) \in \{0 \,..\, b-1\}\right)$
			proveit.numbers.modular.mod_natpos_in_interval
6	instantiation	9, 10, 11	$a \in \mathbb{Z}, b \in \mathbb{Z} \vdash (a+b) \in \mathbb{Z}$
			$a : a, b : b$
7	conjecture		$\vdash 2^t \in \mathbb{N}^+$
			proveit.physics.quantum.QPE._two_pow_t_is_nat_pos
8	axiom		$\vdash \forall_{a,b \in \mathbb{Z}} \left((a \oplus b) = \left((a+b) \bmod 2^t\right)\right)$
			proveit.physics.quantum.QPE._mod_add_def
9	conjecture		$\vdash \forall_{a,b \in \mathbb{Z}} \left((a+b) \in \mathbb{Z}\right)$
			proveit.numbers.addition.add_int_closure_bin
10	assumption		$a \in \mathbb{Z} \vdash a \in \mathbb{Z}$
11	assumption		$b \in \mathbb{Z} \vdash b \in \mathbb{Z}$

FIG. 2. Excerpt from the proof notebook for the `_mod_add_closure` (local QPE) theorem, showing the PROVE-IT-generated formal proof (`In`/`Out` cell 7). The interactive steps leading to this proof are shown in Fig. 1.

eigenvector $|u\rangle$. A Hadamard gate is applied to each line in the first register and then the jth line in the first register controls the application of U^{2^j} to the second register. Then we apply an inverse quantum Fourier transform QFT† on the first register, followed by measurement of the first register, resulting in $|2^t\tilde{\varphi}\rangle$ as the measurement outcome. In our diagrams, we treat measurements as projective such that their output corresponds to the collapsed state as a random variable with the appropriate probability distribution. When the t qubits of the first register are sufficient to represent the phase φ exactly, the QPE algorithm produces $\tilde{\varphi} = \varphi$ and thus φ can be determined from the final measurement $|2^t\varphi\rangle$ by dividing the measurement by 2^t. In the more general case, the estimate $\tilde{\varphi}$ of the actual phase φ can be shown to be a good estimate with high probability if t is sufficiently large.

As suggested, then, by the algorithmic circuit in Fig. 3, we let $t \in \mathbb{N}^+$ and $s \in \mathbb{N}^+$ represent the number of qubits in the first and second registers, respectively. We let $U \in \mathrm{U}(2^s)$ be a unitary matrix of size $2^s \times 2^s$ and $|u\rangle \in \mathbb{C}^{2^s}$ be a unit vector such that $U|u\rangle = e^{2\pi i \varphi}|u\rangle$. These initial assumptions are captured in the following:[2]

$$\text{QPE assumption } \texttt{\_t\_in\_nat\_pos}, \quad t \in \mathbb{N}^+, \tag{2}$$

[2]All such assumptions, definitions, axioms, and theorems are available for in-context inspection at a version of the PROVE-IT website archived specifically for this paper [37]. For readers' convenience, in the online version of this paper, as well as electronic versions of its PDF version, all such axiom and theorem labels in the document are hyperlinked to their specific locations within that archived version of

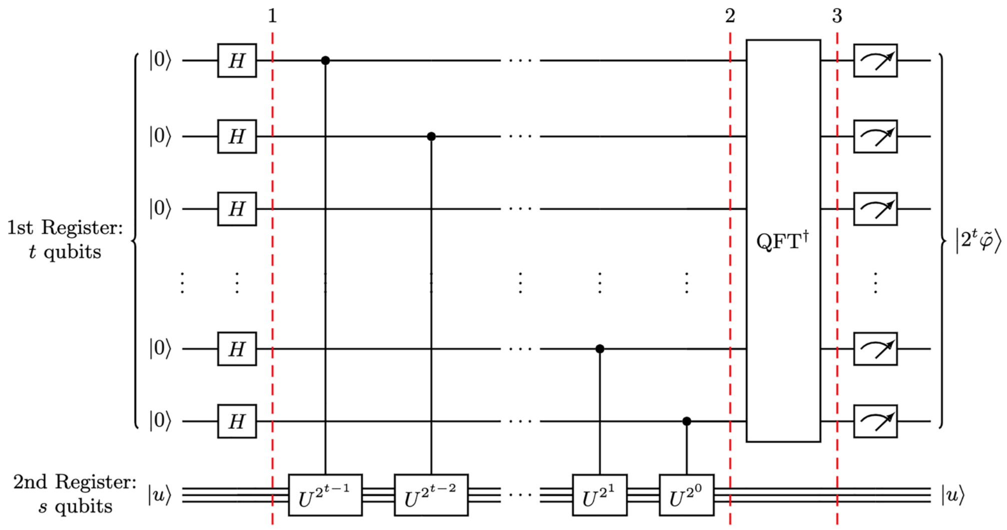


FIG. 3. Quantum circuit implementing the QPE algorithm. From initial first-register state $|0\rangle_t$ and second-register state $|u\rangle$, we apply Hadamard gates to the first register lines, controlled-U^{2^j} gates to the second register, and an inverse quantum Fourier transform QFT† to the first register, leading to first-register measurement $|2^t\tilde{\varphi}\rangle$, where $\tilde{\varphi}$ is an estimate of the phase.

$$\text{QPE assumption \_s\_in\_nat\_pos}, \quad s \in \mathbb{N}^+, \tag{3}$$

$$\text{QPE assumption \_unitary\_U}, \quad U \in \mathrm{U}(2^s), \tag{4}$$

$$\text{QPE assumption \_u\_ket\_register}, \quad |u\rangle \in \mathbb{C}^{2^s}, \tag{5}$$

$$\text{QPE assumption \_normalized\_ket\_u}, \quad \||u\rangle\| = 1, \tag{6}$$

$$\text{QPE assumption \_phase\_in\_interval}, \quad \varphi \in [0, 1), \tag{7}$$

$$\text{QPE assumption \_eigen\_uu}, \quad U|u\rangle = e^{2\pi i\varphi}|u\rangle. \tag{8}$$

Labels beginning with an underscore indicate local axioms or theorems in PROVE-IT, valid only inside a particular package. We would not want, for example, the assumption $t \in \mathbb{N}^+$ to bleed over into other packages. Also note that descriptors such as assumption, definition, or lemma are adopted here for narrative purposes, but PROVE-IT makes no such distinctions (assumptions are local axioms, definitions are axioms, and lemmas are theorems).

We also define a specific modular addition which will be useful in the statements of several lemmas:

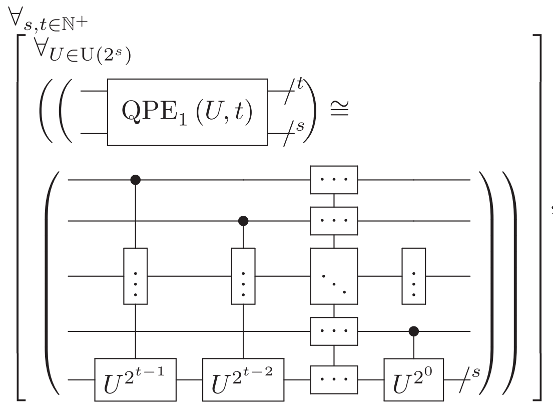
local QPE definition `_mod_add_def`,

$$\forall_{a,b\in\mathbb{Z}}, \quad (a \oplus b) = [(a + b) \bmod 2^t], \tag{9}$$

where $t \in \mathbb{N}^+$ is again the number of qubits available in the first register. This definition is local because it is defined for a problem-specific value of t.

For convenience, we define some simplified representations of pieces of the QPE algorithmic circuit shown in Fig. 3. We represent the portion of the circuit between slices 1 and 2 as the gate QPE$_1(U, t)$, defined with

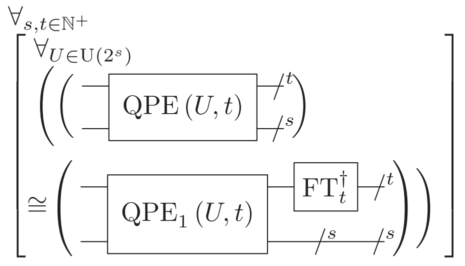
QPE Definition: `QPE1_def`,



, (10)

and the portion of the circuit between slices 1 and 3 as the gate QPE(U, t), defined with

QPE definition `QPE_def`,



, (11)

where the symbol $\cong$ denotes circuit equivalence. Notice that the circuit equivalence `QPE_def` (11) utilizes QPE$_1(U, t)$ on its right-hand side. Notice also that `QPE1_def` (10) and `QPE_def` (11) are definitions of the circuit components and do not include the inputs to, or outputs of, each circuit. Moreover, these circuit components do not include the Hadamard gates. Instead, as illustrated presently, we combine the standard

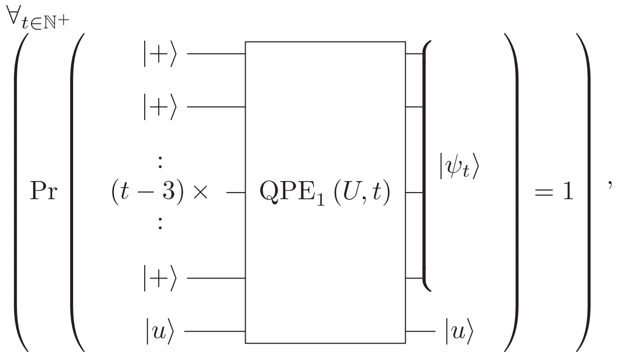
the PROVE-IT website. The most current version of PROVE-IT can be accessed at its public website [33].

initial first-register state $|0\rangle_t$ with the subsequent Hadamard gates to consider the t-qubit state $|+\rangle \otimes \cdots \otimes |+\rangle$ as the input to the QPE1 component of the QPE circuit. The QPE1_def and QPE_def axiomatic definitions are global or universal, valid outside of PROVE-IT's QPE package. The definitions are quantified over $s \in \mathbb{N}^+$, $t \in \mathbb{N}^+$, and $U \in \mathrm{U}(2^s)$ (as variables) instead of holding for the problem-specific s, t, and U values (as literals).

A. QPE circuit state at stage 2

We now prove that the state of the first register at stage 2 is $|\psi_t\rangle$,

local QPE theorem _psi_t_output,



(12)

where $|\psi_t\rangle$ is defined (locally) as

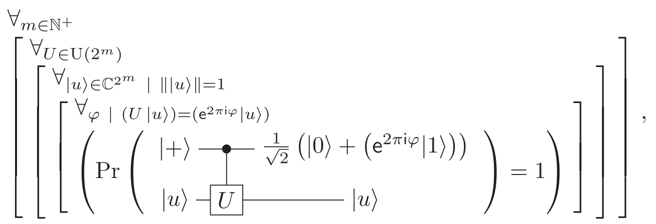
local QPE definition _psi_t_def,

$$\forall_{t\in\mathbb{N}},\quad |\psi_t\rangle = \frac{1}{2^{t/2}}(|0\rangle + e^{2\pi i(2^{t-1}\varphi)}|1\rangle) \otimes (|0\rangle + e^{2\pi i(2^{t-2}\varphi)}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i(2^{0}\varphi)}|1\rangle). \tag{13}$$

Although _psi_t_def (13) quantifies over t as a variable, it is local because it relies upon a problem-specific φ. The theorem _psi_t_output (12) is also local, implicitly relying upon problem-specific assumptions and definitions. The Pr in _psi_t_output (12) and other diagrams below denotes a probability. The theorem _psi_t_output combined with _psi_t_def corresponds to the right-hand side of Nielsen and Chuang's circuit in their Fig. 5.2 and the left-hand side of their Eq. (5.20) and is expressed as a probability (of 1) of obtaining the first-register state $|\psi_t\rangle$ and second-register state $|u\rangle$ from the given inputs to the first stage of the QPE circuit. The probability notation may seem odd here, when we are simply making a claim about the deterministic output of a circuit, but quantum theory is intrinsically probabilistic and we will see how this more general framework is useful later.

The proof of _psi_t_output (12) relies on the more fundamental phase kickback phenomenon, captured in PROVE-IT's quantum circuit package theorems as

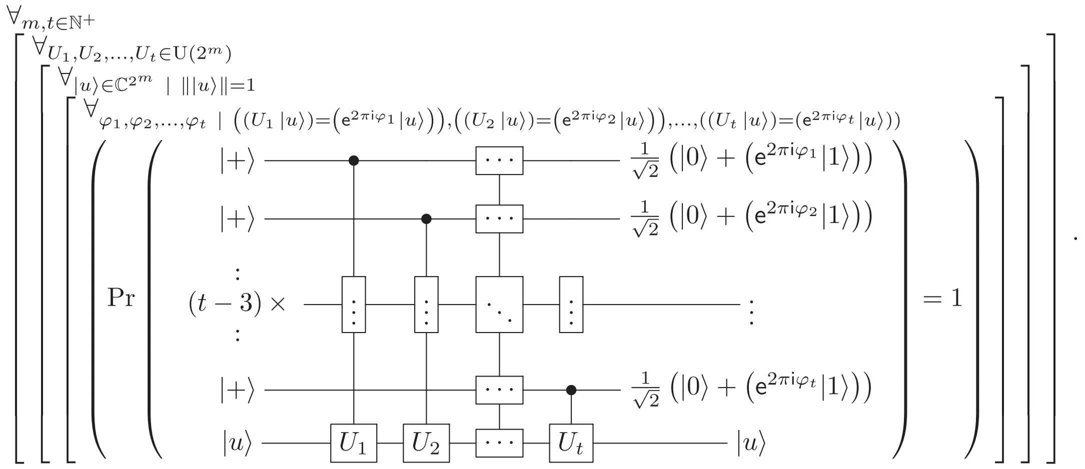
quantum.circuit theorem phase_kickback,


$\forall_{m\in\mathbb{N}^+}$
$\forall_{U\in\mathrm{U}(2^m)}$
$\forall_{|u\rangle\in\mathbb{C}^{2^m}\ |\ \||u\rangle\|=1}$
$\forall_{\varphi\ |\ (U|u\rangle)=(\mathrm{e}^{2\pi i\varphi}|u\rangle)}$
Pr
$|+\rangle$
$\frac{1}{\sqrt{2}}(|0\rangle+(\mathrm{e}^{2\pi i\varphi}|1\rangle))$
$|u\rangle$
U
$|u\rangle$
$=1$

(14)

and generalized for application on a register of multiple qubits as

quantum.circuit theorem phase_kickbacks_on_register,


$\forall_{m,t\in\mathbb{N}^+}$
$\forall_{U_1,U_2,\ldots,U_t\in\mathrm{U}(2^m)}$
$\forall_{|u\rangle\in\mathbb{C}^{2^m}\ |\ \||u\rangle\|=1}$
$\forall_{\varphi_1,\varphi_2,\ldots,\varphi_t\ |\ ((U_1|u\rangle)=(\mathrm{e}^{2\pi i\varphi_1}|u\rangle)),((U_2|u\rangle)=(\mathrm{e}^{2\pi i\varphi_2}|u\rangle)),\ldots,((U_t|u\rangle)=(\mathrm{e}^{2\pi i\varphi_t}|u\rangle))}$
Pr
$|+\rangle$
$\frac{1}{\sqrt{2}}(|0\rangle+(\mathrm{e}^{2\pi i\varphi_1}|1\rangle))$
$|+\rangle$
$\frac{1}{\sqrt{2}}(|0\rangle+(\mathrm{e}^{2\pi i\varphi_2}|1\rangle))$
$(t-3)\times$
$|+\rangle$
$\frac{1}{\sqrt{2}}(|0\rangle+(\mathrm{e}^{2\pi i\varphi_t}|1\rangle))$
$|u\rangle$
U_1
U_2
U_t
$|u\rangle$
$=1$

(15)

We then prove that the tensor product in _psi_t_def (13) can be rewritten in the compact summation form corresponding to Nielsen and Chuang's equation (5.20) (see [31], p. 222)[3]

[3]Here $|k\rangle_t$ denotes a number ket (a NumKet object in PROVE-IT), similar to notation used by Mermin [38] for a multiple-Cbit state, where k is the decimal equivalent of a binary sequence within the ket and the subscript t indicates the number of qubits being used in the representation. The vector $|11010\rangle$ on five qubits, for example, can be represented by the number ket $|26\rangle_5$.

local QPE theorem `_psi_t_formula`,

$$\forall_{t\in\mathbb{N}}, \quad |\psi_t\rangle = \frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}|k\rangle_t. \tag{16}$$

The `_psi_t_formula` theorem (16) is notable for using a proof by induction (on the variable t representing the number of qubits in the first register). For some insight into the transformation from the tensor product in `_psi_t_def` (13) to the summation formula in `_psi_t_formula` (16), consider the instantiation and expansion of `_psi_t_def` for $t = 1$ and 2 shown below:

$$|\psi_1\rangle = \frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i\varphi(1)}|1\rangle) = \frac{1}{2^{1/2}}\sum_{k=0}^{1} e^{2\pi i\varphi k}|k\rangle_t, \tag{17}$$

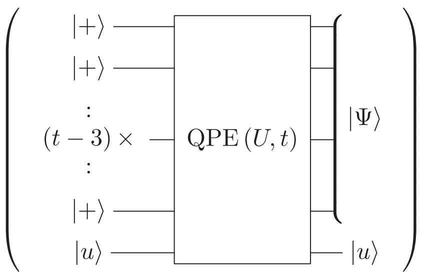
$$\begin{aligned}
|\psi_2\rangle &= \frac{1}{2^{2/2}}((|0\rangle + e^{2\pi i\varphi(2)}|1\rangle)\otimes(|0\rangle + e^{2\pi i\varphi(1)}|1\rangle))\\
&= \frac{1}{2}(|00\rangle + e^{2\pi i\varphi(1)}|01\rangle + e^{2\pi i\varphi(2)}|10\rangle + e^{2\pi i\varphi(3)}|11\rangle)\\
&= \frac{1}{2}(|0\rangle_2 + e^{2\pi i\varphi(1)}|1\rangle_2 + e^{2\pi i\varphi(2)}|2\rangle_2 + e^{2\pi i\varphi(3)}|3\rangle_2)\\
&= \frac{1}{2}\sum_{k=0}^{3} e^{2\pi i\varphi k}|k\rangle_t.
\end{aligned} \tag{18}$$

B. QPE circuit state at stage 3

We next prove that the first-register state at stage 3 is $|\Psi\rangle$,

local QPE theorem `_Psi_output`,

Pr



$= 1, \quad (19)$

where $|\Psi\rangle$ is defined locally from `_Psi_def` as

local QPE definition `_Psi_def`,

$$|\Psi\rangle = \mathrm{FT}_t^{\dagger}|\psi_t\rangle, \tag{20}$$

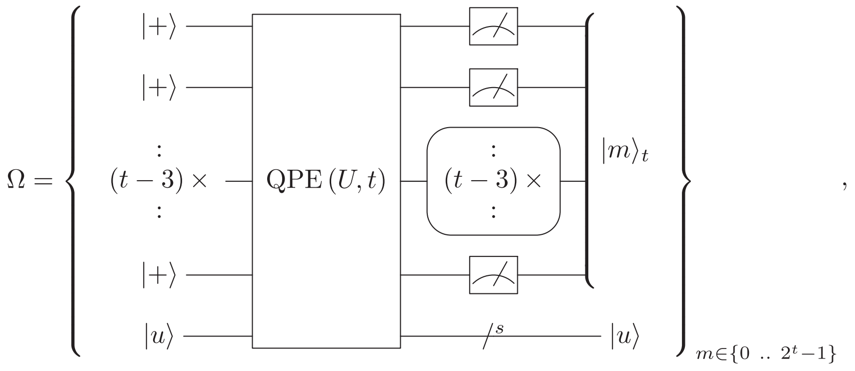
where $\mathrm{FT}_t^{\dagger}$ represents a t-qubit inverse quantum Fourier transform, and Pr again denotes a probability. This follows trivially from the definition (11) of the QPE circuit. The inverse quantum Fourier transform is an important quantum algorithm of its own, of course, but is treated here as a modular black box function returning a well-defined result.

As with `_psi_t_output` (12), `_Psi_output` (19) is expressed as a probability, a probability of 1 that we obtain at stage 3 the first-register state $|\Psi\rangle$ and second-register state $|u\rangle$ from the given inputs.

C. Probability space of measurement outcomes

To carefully formalize the QPE process and related measurement probabilities, we take some care in establishing a probability space to formally model the QPE circuit measurement outcomes. Interpreting the sequence of 0's and 1's obtained in the measurement process as the binary expansion of an integer m, we define the sample space Ω as the set of all possible QPE circuit configurations with the given fixed input state $|+\rangle \otimes |+\rangle \otimes \cdots \otimes |+\rangle \otimes |u\rangle$ and (probabilistic) first-register measurement result $|m\rangle_t$, with $m \in \{0, 1, \dots 2^t - 1\}$,

local QPE definition `_sample_space_def`,

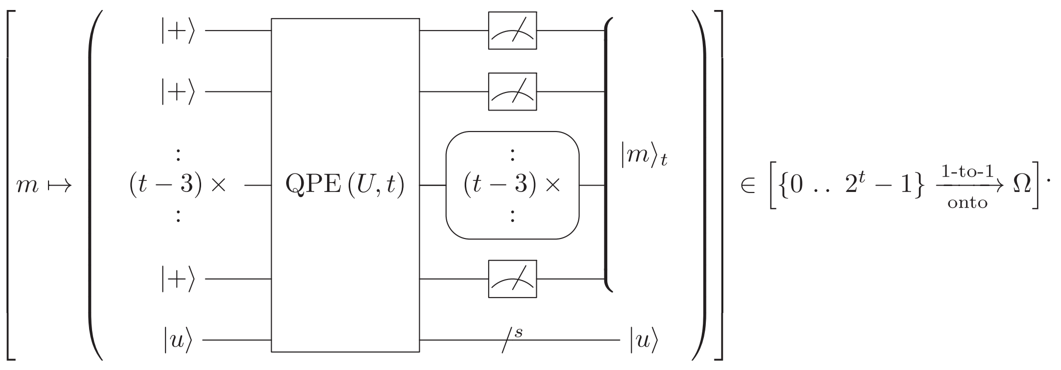


(21)

where the right-hand side of `_sample_space_def` (21) is an example of a set comprehension expression in PROVE-IT, i.e., the right-hand side of `_sample_space_def` denotes a set of input-output-circuit configurations and the large curly braces are set delimiters.[4]

To prove that Ω has the properties of a sample space, we need to establish that the mapping from integer m to the QPE circuit configuration with measurement $|m\rangle_t$ is a one-to-one mapping from the set $\{0, 1, 2, \dots, 2^t - 1\}$ onto Ω,

local QPE theorem `_sample_space_bijection`,

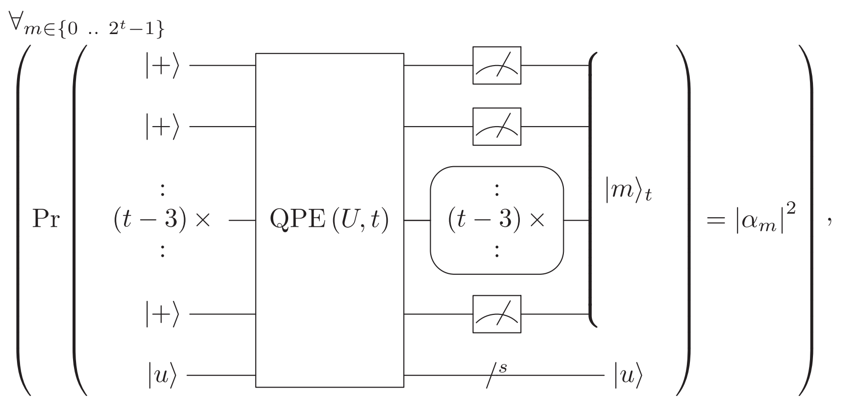


(22)

This means that the values $m \in \{0, 1, 2, \dots, 2^t - 1\}$ induce a partition of the sample space Ω, which is true given that quantum circuits with distinct outputs are distinct from each other. This is also critical later in evaluating the probability of an event consisting of the disjunction $\{|m_1\rangle_t, |m_2\rangle_t, \dots, |m_k\rangle_t\}$ of atomic outcomes, allowing us to use the fact that $\Pr(\bigvee_i |m_i\rangle_t) = \sum_i \Pr(|m_i\rangle_t)$.

The probability function associated with the probability space is then obtained as a special case of the Born rule (recall above we established $|\Psi\rangle$ as the first-register state of the QPE circuit at stage 3, just before measurement),

local QPE theorem `_outcome_prob`,



(23)

[4]Set comprehension expressions in PROVE-IT look pretty much like standard set comprehension expressions except that domain specifications appear at the end in subscript form. For example, a (sub)set comprehension expression in PROVE-IT for the positive Reals might appear as $\{x\,|\,x > 0\}_{x\in\mathbb{R}}$.

where α_m is the probability amplitude corresponding to the measurement outcome $|m\rangle$,

$$\begin{aligned}&\text{local QPE definition } \texttt{\_alpha\_m\_def},\\ &\forall_{m\in\{0,\dots,2^t-1\}}(\alpha_m = \langle m|\Psi\rangle).\end{aligned} \tag{24}$$

We can then prove that the set Ω defined in `_sample_space_def` (21) is indeed a sample space,

$$\begin{aligned}&\text{local QPE theorem } \texttt{\_Omega\_is\_sample\_space},\\ &\Omega \underset{c}{\in} \text{SampleSpaces},\end{aligned} \tag{25}$$

meaning that the outcomes specified in Ω are mutually exclusive and collectively exhaustive, which then also implies that the probabilities of the individual outcomes in Ω sum to 1.[5]

D. Evaluating probability amplitude α_m

We then prove an initial formula for α_m. As Nielsen and Chuang [see [31], p. 223, formula (5.23)] point out, applying the inverse Fourier transform to the state in `_psi_t_formula` (16) produces the double-summation state

$$|\Psi\rangle = \frac{1}{2^t}\sum_{k,l=0}^{2^t-1} e^{-2\pi i k\ell/2^t} e^{2\pi i\varphi k}|\ell\rangle. \tag{26}$$

From the definition of α_m in `_alpha_m_def` (24) and `_psi_t_formula` (16), we can derive a formula for the probability amplitude corresponding to measurement outcome $|m\rangle$,

$$\begin{aligned}&\text{local QPE theorem } \texttt{\_alpha\_m\_evaluation},\\ &\forall_{m\in\{0,\dots,2^t-1\}},\\ &\alpha_m = \left(\frac{1}{2^t}\sum_{k=0}^{2^t-1}(e^{-2\pi i k m/2^t} e^{2\pi i\varphi k})\right).\end{aligned} \tag{27}$$

This follows straightforwardly from the definitions of $|\Psi\rangle$ (13), $|\psi_t\rangle$ (20), and

$$\begin{aligned}&\text{QFT theorem } \texttt{invFT\_on\_matrix\_elem},\\ &\forall_{n\in\mathbb{N}^+},\quad \forall_{k,l\in\{0\cdots 2^n-1\}},\\ &({}_n\langle l|\text{FT}_n^\dagger|k\rangle_n) = \left(\frac{1}{2^{n/2}}e^{-2\pi i k l/2^n}\right).\end{aligned} \tag{28}$$

The formula `_alpha_m_evaluation` (27) makes explicit an implicit step in the work of Nielsen and Chuang, intermediate between their expression (5.23) and formula (5.24) (see [31], p. 223).

E. Special case: The phase φ can be represented exactly in t bits

For the special case when the first register of t qubits in the QPE circuit in Fig. 3 is sufficient to exactly represent the phase φ [called the ideal case by Nielsen and Chuang (see [31], p. 223)], the ideal phase condition is given by

$$2^t\varphi \in \{0, 1, 2, \dots, 2^t - 1\}. \tag{29}$$

Instantiating `_alpha_m_evaluation` (27) with $m = 2^t\varphi$ under this condition leads to

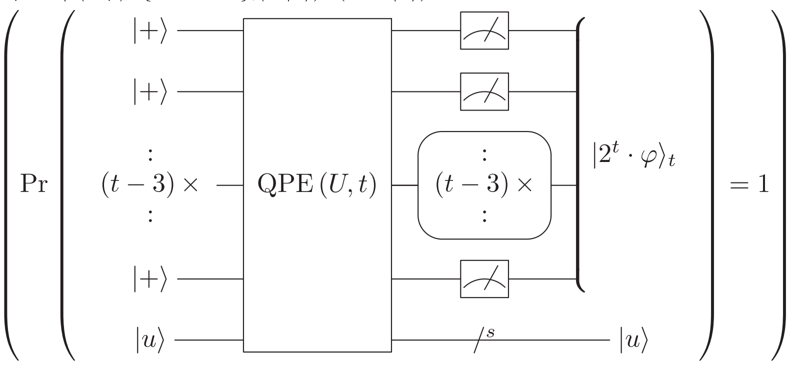
$$\begin{aligned}&\text{local QPE theorem } \texttt{\_alpha\_ideal\_case},\\ &[(2^t\varphi) \in \{0, 1, 2, \dots, 2^t - 1\}] \Rightarrow (\alpha_{(2^t\varphi)} = 1),\end{aligned} \tag{30}$$

[5]The $\underset{c}{\in}$ notation with the underset c is used to indicate class membership and distinguish it from standard set membership. Here we are simply taking the collection of all sample spaces to be a class, which is weaker than assuming the collection actually constitutes a set (only sets admit subset comprehension).

that is, the probability amplitude of the scaled phase $2^t\varphi$ is 1. Using `_outcome_prob` (23) and `_alpha_ideal_case` (30), we obtain

QPE theorem `qpe_exact`,

$$\forall_{s,t\in\mathbb{N}+}\Big[\forall_{U\in\mathrm{U}(2^s)}\Big[\forall_{|u\rangle\in\mathbb{C}^{2^s}\ |\ \||u\rangle\|=1}\Big[\forall_{\varphi\in\mathbb{R}\ |\ (2^t\varphi)\in\{0\ ..\ 2^t-1\},(U\,|u\rangle)=(\mathrm{e}^{2\pi\mathrm{i}\varphi}\,|u\rangle)}\Big[\ \cdots\ \Big]\Big]\Big]\Big], \tag{31}$$



which states that, under the standard QPE problem conditions and under the specific additional ideal phase condition that $2^t\varphi \in \{0, \ldots, 2^t - 1\}$ (i.e., that the phase φ can be written exactly with a t-bit binary expansion), the first-register measurement in the QPE circuit gives exactly $|2^t\varphi\rangle_t$ with a probability of 1. The phase φ can then be obtained by dividing the measurement result by 2^t.

F. Bounding the probability for the best measurement outcome $b_r = \texttt{round}(2^t\varphi)$

When $2^t\varphi$ cannot be represented exactly in the t qubits available in the first register, we can still prove a lower bound for the probability of obtaining the best possible measurement outcome, $b_r = \texttt{round}(2^t\varphi)$. From `_alpha_m_evaluation` (27) and using the 2^t periodicity of $e^{2\pi ikm/2^t}$, we can derive

local QPE theorem `_alpha_m_mod_evaluation`,

$$\forall_{m\in\mathbb{Z}},\quad \alpha_{(m \bmod 2^t)} = \left(\frac{1}{2^t}\sum_{k=0}^{2^t-1}\left(e^{-2\pi ikm/2^t}e^{2\pi i\varphi k}\right)\right) \tag{32}$$

and then rewrite this explicitly as a finite geometric sum

local QPE theorem

`_alpha_m_mod_as_geometric_sum`,

$$\forall_{m\in\mathbb{Z}},\quad \alpha_{(m \bmod 2^t)} = \left(\frac{1}{2^t}\sum_{k=0}^{2^t-1}\left(e^{2\pi i(\varphi-m/2^t)}\right)^k\right). \tag{33}$$

We now define b_r and δ_b as follows:

local QPE definition `_best_round_def`, (34)

$$b_r = \texttt{round}(2^t\varphi);$$

local QPE definition `_delta_b_def`, (35)

$$\forall_{b\in\mathbb{Z}},\quad \delta_b = \varphi - \frac{b}{2^t}.$$

Reducing the finite geometric sum of `_alpha_m_mod_as_geometric_sum` (33) and using $\theta > 0 \Rightarrow \sin\theta < \theta$ and $0 \leqslant \theta \leqslant \pi/2 \Rightarrow \sin\theta \geqslant 2\theta/\pi$ (as illustrated in Fig. 4), it follows that

local QPE theorem

`_best_guarantee_delta_nonzero`,

$$(\delta_{b_r} \neq 0) \Rightarrow \left(|\alpha_{b_r \bmod 2^t}|^2 > \frac{4}{\pi^2}\right). \tag{36}$$

Specifically, when the t qubits of the first register are insufficient to exactly represent the scaled phase $2^t\varphi$, i.e., $\delta_{b_r} \neq 0$, the squared norm of the probability amplitude of the rounding-based best estimate b_r is greater than $4/\pi^2$. Of course, this does not really require that $\delta_{b_r} \neq 0$ since `qpe_exact` (31)

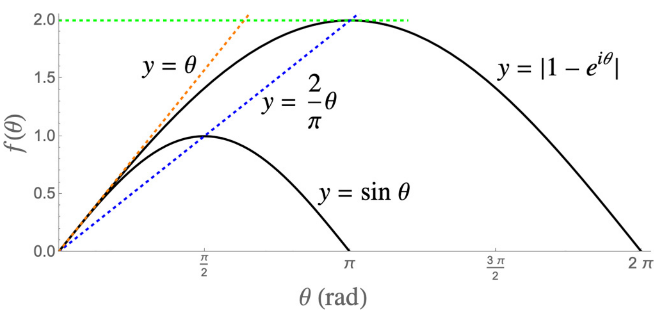


FIG. 4. Illustration of the bounds $\frac{2}{\pi}\theta \leqslant |1 - e^{i\theta}| \leqslant 2$ for $\theta \in [0, \pi]$ and $\sin\theta \geqslant \frac{2}{\pi}\theta$ for $\theta \in [0, \frac{\pi}{2}]$, utilized in the proof of `_alpha_sqrd_upper_bound` (47), and the bound $\sin\theta < \theta$ for $\theta > 0$ used in the proof of `_best_guarantee_delta_nonzero` (36).

proves that the probability is 1 when $\delta_{b_r} = 0$. Therefore,

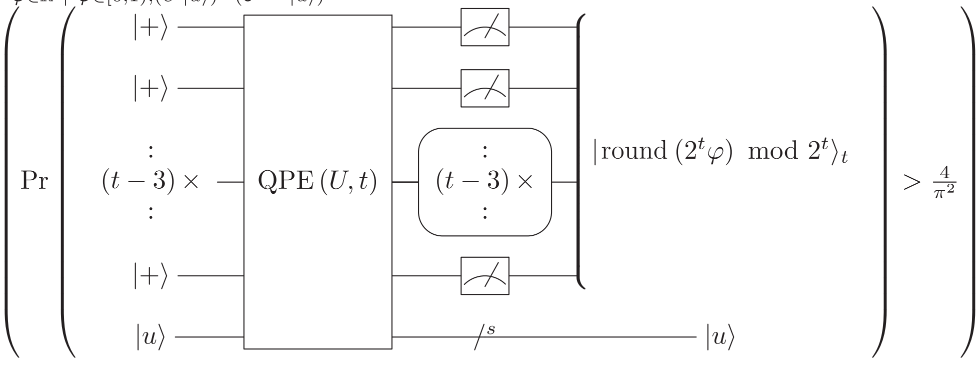
local QPE theorem `_best_guarantee`,

$$|\alpha_{b_r \bmod 2^t}|^2 > \frac{4}{\pi^2}, \tag{37}$$

which then leads to the general `qpe_best_guarantee` theorem, providing a lower bound on the probability of obtaining $b_r = \mathtt{round}(2^t\varphi)$ as the measurement outcome,

QPE theorem `qpe_best_guarantee`,

$$\forall_{s,t\in\mathbb{N}^+} \left[\forall_{U\in\mathrm{U}(2^s)} \left[\forall_{|u\rangle\in\mathbb{C}^{2^s} \;|\; \||u\rangle\|=1} \left[\forall_{\varphi\in\mathbb{R} \;|\; \varphi\in[0,1),(U\,|u\rangle)=(\mathrm{e}^{2\pi\mathrm{i}\varphi}|u\rangle)} \right.\right.\right.$$



$$\left.\left.\left. \right]\right]\right], \tag{38}$$

which is consistent with earlier nonformal results [36,39] as well as recent formal proof results in COQ [40], but they used a weaker (nonstrict) inequality ($\geqslant$).

G. Guaranteeing a desired precision

Closely matching Nielsen and Chuang's (see [31], p. 223) approach in their Sec. 5.2.1 on performance and requirements, let the integer $b_f \in \{0, 1, 2, \dots, 2^t - 1\}$ be such that $b_f/2^t = 0.b_1b_2\cdots b_t$ is the best t-bit approximation to the phase φ which is less than φ; this is expressed as

local QPE definition `_best_floor_def`,

$$b_f = \lfloor 2^t\varphi \rfloor, \tag{39}$$

giving a difference $\delta_{b_f} = \varphi - \frac{b_f}{2^t} > 0$; recall an earlier definition for δ (35) and note that the ideal case in Sec. II E where $b_f = b_r = \varphi$ producing $\delta_{b_f} = 0$ has already been considered.

From the finite geometric sum expression in `_alpha_m_mod_as_geometric_sum` (33), we can prove

local QPE theorem `_alpha_summed`,

$$\forall_{\ell\in\{-2^{t-1}+1,\dots,2^{t-1}\} \,|\, \ell\neq 0},$$

$$\alpha_{b_f\oplus\ell} = \left(\frac{1}{2^t}\,\frac{1-e^{2\pi i(2^t\delta_{b_f}-\ell)}}{1-e^{2\pi i(\delta_{b_f}-\ell/2^t)}}\right), \tag{40}$$

which corresponds to expression (5.26) in Ref. [31] (p. 224) with minor differences in notation and the domain for ℓ now explicitly stated. The theorem `_alpha_summed` (40) gives a formula for the probability amplitude corresponding to a measurement outcome that is ℓ units away from the desired best measurement outcome b_f (39).

Proving `_alpha_summed` (40) involves instantiating `_alpha_m_mod_as_geometric_sum` (33) with $m = b_f + \ell$ to produce the equivalent of Nielsen and Chuang's expression (5.24),

$$\ell \in \{-2^{t-1}+1,\dots,2^{t-1}\}$$

$$\vdash \left(\alpha_{b_f\oplus\ell} = \frac{1}{2^t}\sum_{k=0}^{2^t-1}(e^{2\pi i[\varphi-(b_f+\ell)/2^t]})^k\right), \tag{41}$$

then substituting $\varphi = \delta_{b_f} + \frac{b_f}{2^t}$ and simplifying to obtain

$$\ell \in \{-2^{t-1}+1,\dots,2^{t-1}\}$$

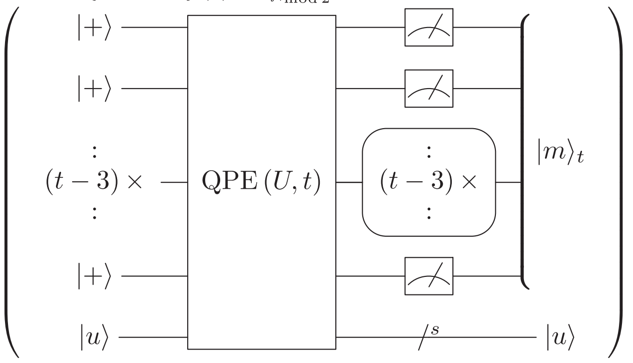
$$\vdash \left(\alpha_{b_f\oplus\ell} = \frac{1}{2^t}\sum_{k=0}^{2^t-1}(e^{2\pi i(\delta_{b_f}-\ell/2^t)})^k\right), \tag{42}$$

and then reducing the geometric sum. Reducing the geometric sum requires that the common ratio $e^{2\pi i(\delta_{b_f}-\ell/2^t)}$ is not 1, which follows from the fact that $(\delta_{b_f} - \frac{\ell}{2^t}) \notin \mathbb{Z}$ unless $\delta_{b_f} = 0$, which was handled as the special case in Sec. II E.

Letting e be "a positive integer characterizing our desired tolerance to error" (see [31], p. 224), we define the probability of a successful measurement outcome (and its complementary failure) with the following definitions:

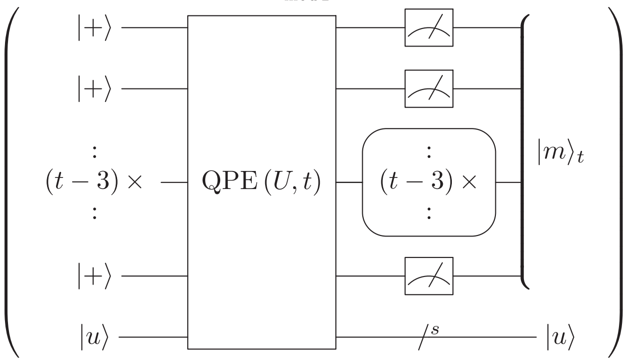
local QPE definition `_success_def`,

$$\forall_{e \in \{1\ ..\ 2^{t-1}-2\}} \left([P_{\text{success}}](e) = \left[\text{Prob}_{m \in \{0\ ..\ 2^t-1\}\ |\ |m-b_f|_{\text{mod}\ 2^t} \leq e} \left(\ \right) \right] \right); \tag{43}$$



local QPE definition `_fail_def`,

$$\forall_{e \in \{1\ ..\ 2^{t-1}-2\}} \left([P_{\text{fail}}](e) = \left[\text{Prob}_{m \in \{0\ ..\ 2^t-1\}\ |\ |m-b_f|_{\text{mod}\ 2^t} > e} \left(\ \right) \right] \right). \tag{44}$$



The definition `_success_def` (43) states that the probability $[P_{\text{success}}](e)$ of a measurement m that is no more than e units away from the best underestimate b_f of $2^t\varphi$ is simply the probability[6] of obtaining the QPE circuit configuration with first-register measurement output $|m\rangle_t$ such that $|m - b_f|_{\text{mod}\, 2^t} \leqslant e$. The probability of failing to obtain a measurement within the desired error tolerance e of b_f is defined similarly, but now with the condition that $|m - b_f|_{\text{mod}\, 2^t} > e$. Nielsen and Chuang express the failure probability using the notation $p(|m - b| > e)$ [see [31], p. 224, left-hand side of Eq. (5.27)]. Although the constraint is unstated by Nielsen and Chuang, the upper bound on e of $2^{t-1} - 1$ is important to ensure that the summations that are to imminently appear in `_fail_sum` (45) are properly defined.

Recall from earlier that $\alpha_{b_f \oplus \ell}$ is the probability magnitude of the vector $|b_f \oplus \ell\rangle_t$, where b_f is the best t-bit underestimate for the scaled phase $2^t\varphi$. We also defined the tolerable error e as the number of units away from the best estimate b that would be acceptable as an eventual measurement. The probability, then, of failing to obtain a sufficiently accurate estimate of the phase φ is the sum of the squared magnitudes of all possible measurements $|m\rangle_t$ such that $|m - b_f| > e$. This is proven in

local QPE theorem `_fail_sum`,

$$\forall_{e \in \{1 \ldots 2^{t-1}-2\}},$$
$$[P_{\text{fail}}](e) = \left(\sum_{\ell=-2^{t-1}+1}^{-(e+1)} |\alpha_{b_f \oplus \ell}|^2 \right) + \left(\sum_{\ell=e+1}^{2^{t-1}} |\alpha_{b_f \oplus \ell}|^2 \right), \tag{45}$$

[6]The $\text{Prob}_{\text{conds}}(E)$ in (43) and (44) denotes the probability of an event E specified by the condition conds.

which corresponds to formula (5.27) in Ref. [31] (p. 224). Part of proving (45) involves showing that the $|m - b_f|_{\text{mod } 2^t} > e$ condition in `_fail_def` (44) is consistent with the ranges of the two summations over ℓ (which takes the role of $m - b_f$):

local QPE theorem `_fail_sum_prob_conds_equiv_lemma`,

$$\forall_t \left[\begin{array}{l} \forall_{e \in \{1 \cdots 2^{t-1}-2\}}, \\ [(l \in [\{-2^{t-1}+1 \cdots -(e+1)\} \cup \{e+1 \cdots 2^{t-1}\}]) \wedge (|l|_{\text{mod } 2^t} > e)] \\ \qquad \Leftrightarrow [(l \in \{-2^{t-1}+1 \cdots 2^{t-1}\}) \wedge (|l|_{\text{mod } 2^t} > e)] \end{array}\right]. \tag{46}$$

From `_alpha_summed` (40), we can then establish an upper bound on $|\alpha_{b_f \oplus \ell}|^2$,

local QPE theorem `_alpha_sqrd_upper_bound`,

$$\forall_{l \in \{-2^{t-1}+1, \ldots, 2^{t-1}\}}, \quad |\alpha_{b_f \oplus \ell}|^2 \leqslant \frac{1}{4[\ell - (2^t \delta_{b_f})]^2}, \tag{47}$$

corresponding to formula (5.29) in Ref. [31] (p. 224) and making use of the bounds $\frac{2}{\pi}\theta \leqslant |1 - e^{i\theta}| \leqslant 2$ for $\theta \in [0, \pi]$ (as illustrated in Fig. 4).

From `_fail_sum` (45) and `_alpha_sqrd_upper_bound` (47) we then derive an initial upper bound expression for $[P_{\text{fail}}](e)$,

local QPE theorem `_failure_upper_bound_lemma`,

$$\forall_{e \in \{1, \ldots, 2^{t-1}-2\}}, \quad [P_{\text{fail}}](e) \leqslant \frac{1}{4}\left(\sum_{\ell=-2^{t-1}+1}^{-(e+1)} \frac{1}{[\ell - (2^t\delta)]^2} + \sum_{\ell=e+1}^{2^{t-1}} \frac{1}{[\ell - (2^t\delta)]^2}\right), \tag{48}$$

corresponding to bound (5.30) in Ref. [31] (p. 224). Applying some standard summation manipulations and inequalities to `_failure_upper_bound_lemma` (48), we then prove

local QPE theorem `_failure_upper_bound`,

$$\forall_{e \in \{1 \cdots 2^{t-1}-2\}}, \quad [P_{\text{fail}}](e) \leqslant \frac{1}{2e} + \frac{1}{4e^2}, \tag{49}$$

corresponding to bound (5.34) in Ref. [31] (p. 224), but actually a slight improvement over their bound in two different ways: First, the bound holds for $e \geqslant 1$ whereas their original bound holds only for $e \geqslant 2$, and second, the bound in `_failure_upper_bound` (49) is actually tighter (i.e., less) than Nielsen and Chuang's bound and thus the eventually derived probability for a successful measurement is actually larger. See Fig. 5 for a graphical comparison of the original and new bounds.

From `_success_def` (43) and `_fail_def` (44), we trivially derive

local QPE theorem `_success_complements_failure`,

$$\forall_{e \in \{1 \cdots 2^{t-1}-2\}}, \quad [P_{\text{success}}](e) = 1 - [P_{\text{fail}}](e). \tag{50}$$

We can then show that adopting Nielsen and Chuang's formula (5.35) (see [31], p. 224) as an assumed lower bound on the number of qubits in the first register,

QPE assumption `_t_req`,

$$t \geqslant n + \left\lceil \log_2\left(2 + \frac{1}{2\epsilon}\right)\right\rceil, \tag{51}$$

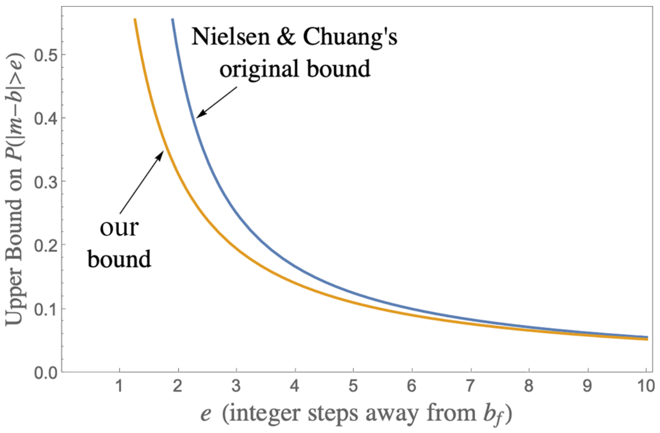
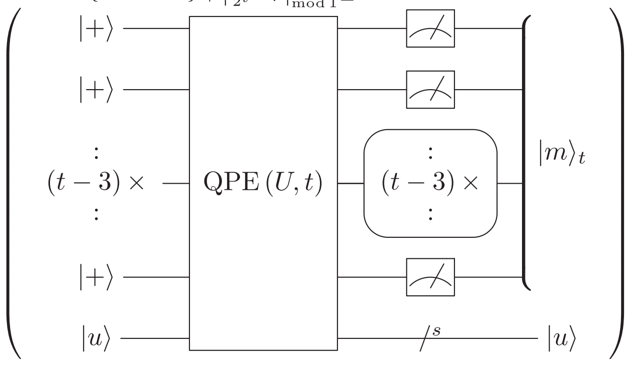


FIG. 5. Comparison of Nielsen and Chuang's [31] original upper bound vs our tighter upper bound on the probability of failure versus the number of unit steps e away from the best floor-based estimate b_f. See the `_failure_upper_bound` theorem (49).

is sufficient to guarantee a successful measurement of phase φ to n bits of precision with a probability of at least $1-\epsilon$:

local QPE theorem `_precision_guarantee`,

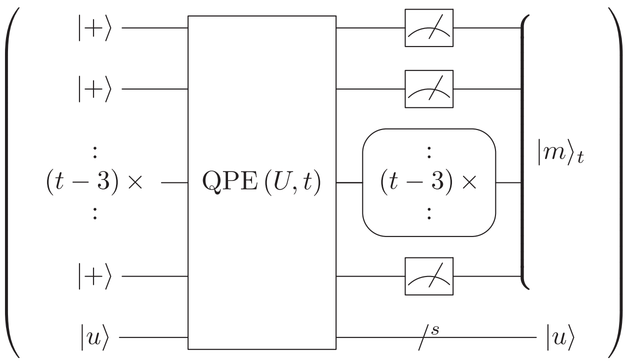
$$\left[\text{Prob}_{m \in \{0\ ..\ 2^t-1\}\ \mid\ \left|\frac{m}{2^t}-\varphi\right|_{\text{mod }1} \leq 2^{-n}}\left(\cdots\right)\right] \geq (1-\epsilon)\,. \tag{52}$$



That local theorem is then generalized (e.g., over all failure probabilities $\epsilon \in (0, 1]$ and all conditions satisfying the original QPE problem statement) to produce the universal `qpe_precision_guarantee` theorem

QPE theorem `qpe_precision_guarantee`,

$$\forall_{\epsilon \in (0,1]}\left[\forall_{s,n \in \mathbb{N}^+\ \mid\ n \geq 2}\left[\forall_{U \in \mathrm{U}(2^s), |u\rangle \in \mathbb{C}^{2^s}, \varphi \in \mathbb{R}\ \mid\ \varphi \in [0,1), \||u\rangle\|=1, (U\,|u\rangle) = (\mathrm{e}^{2\pi \mathrm{i}\varphi}\,|u\rangle)}\left[\forall_{t \in \mathbb{N}^+\ \mid\ t \geq \left(n + \left\lceil \log_2\left(2+\frac{1}{2\epsilon}\right)\right\rceil\right)}\left(\left[\text{Prob}_{m \in \{0\ ..\ 2^t-1\}\ \mid\ \left|\frac{m}{2^t}-\varphi\right|_{\text{mod }1} \leq 2^{-n}}\left(\cdots\right)\right] \geq (1-\epsilon)\right)\right]\right]\right]. \tag{53}$$

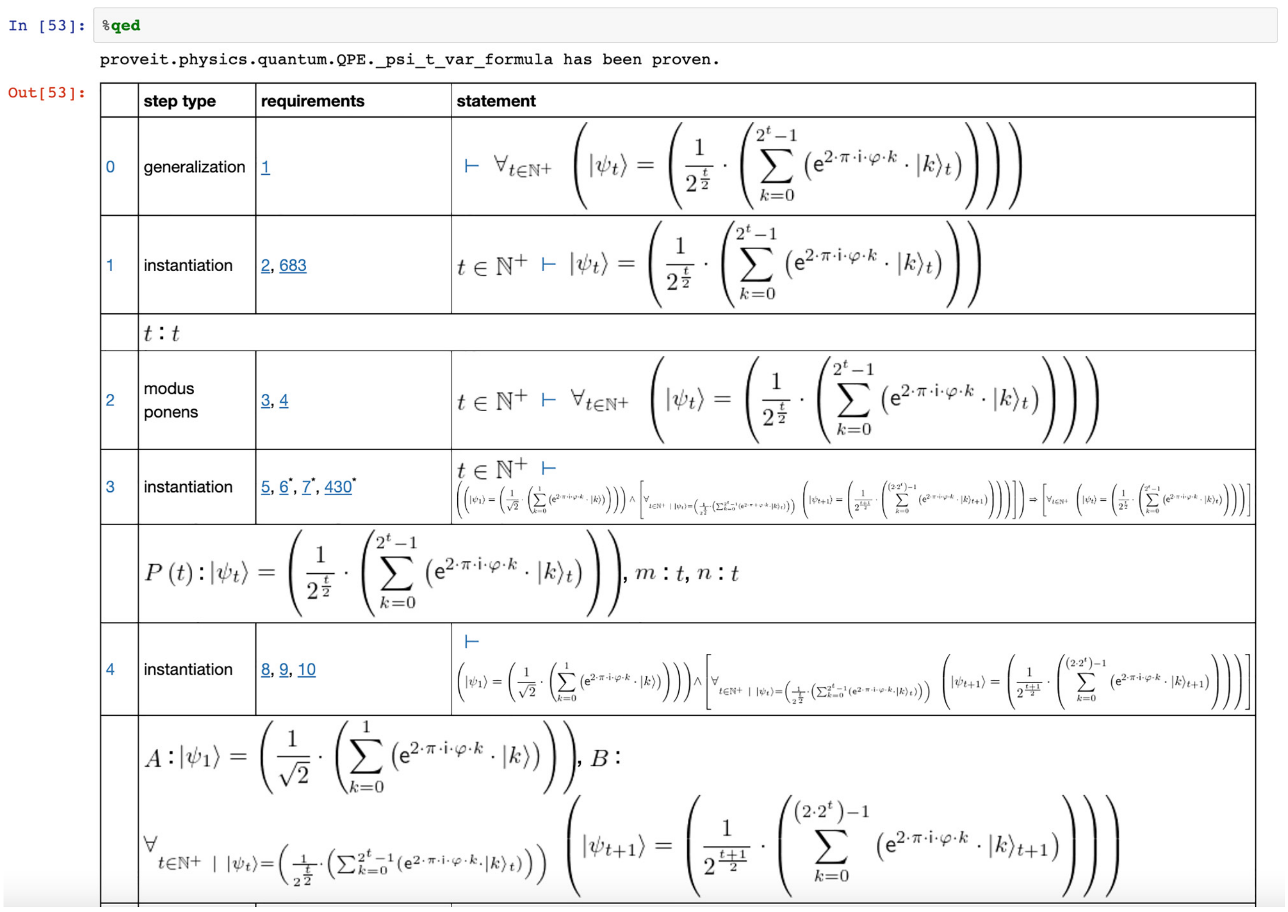


III. WHAT WE PRECISELY PROVED

Using our PROVE-IT system, we have constructed formal proofs of three universal theorems pertaining to quantum phase estimation: `qpe_exact` (31), `qpe_best_guarantee` (38), and `qpe_precision_guarantee` (53). Those statements, as well as an outline of the steps leading to their derivation, are shown in Sec. II. We address the following questions: What does it mean to say that we have these formal proofs? What are the precise claims represented by such proofs and how are these claims verified?

A. Formal proofs in PROVE-IT

A formal proof or proof certificate in PROVE-IT is a finite sequence $A_1, A_2, \ldots, A_n$ of formulas, each of which is either an assumption, an axiom, a theorem (proven or unproven), or a derivation consisting of a judgment using a rule of inference whose conditions appear among the previous formulas and ending with A_n as the theorem being proved [41,42]. This is essentially the same as the definition of a formal proof in mathematical logic except for the flexibility offered by PROVE-IT to treat proven or unproven theorems effectively as axioms at the local level while tracking their dependencies and protecting against circular logic on a broader level. The judgment in each step has the form $\Gamma \vdash A$, indicating that A can be derived within the system assuming the (possibly empty) set of additional assumptions Γ (consistent with notation in modern treatments of mathematical logic such as [41] using Frege's assertion sign or turnstile symbol $\vdash$ as discussed in [43]). The rules of inference include *modus ponens*, deduction, instantiation, and generalization (described in detail in Appendix C of [32]). A formal proof in PROVE-IT also includes several helpful characteristics of a structured proof [44,45], including rules of inference and previous-step dependencies being explicitly cited in each step and hyperlinks allowing the user to isolate, zoom into, and explore the details of any step.



```
In [53]: %qed
```

```
proveit.physics.quantum.QPE._psi_t_var_formula has been proven.
```

Out[53]:

	step type	requirements	statement
0	generalization	1	$\vdash \forall_{t \in \mathbb{N}^+} \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)\right)$
1	instantiation	2, 683	$t \in \mathbb{N}^+ \vdash \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)$
	$t : t$		
2	modus ponens	3, 4	$t \in \mathbb{N}^+ \vdash \forall_{t \in \mathbb{N}^+} \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)\right)$
3	instantiation	5, 6*, 7*, 430*	$t \in \mathbb{N}^+ \vdash \left(\left(\lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle\right)\right)\right)\right) \wedge \left[\forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)\right]\right) \Rightarrow \left[\forall_{t \in \mathbb{N}^+} \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)\right)\right]$
	$P(t) : \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right), m : t, n : t$		
4	instantiation	8, 9, 10	$\vdash \left(\lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle\right)\right)\right)\right) \wedge \left[\forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)\right]$
	$A : \lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle\right)\right)\right), B : \forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)$		

FIG. 6. First few lines of PROVE-IT's formal proof output for _psi_t_formula (16), illustrating the general structure of a formal proof in PROVE-IT and the rigorous and explicitly human-checkable details involved in each proof step. Each numbered step (numbers along the left-hand side) includes an explicit listing (underlined numbers in the "requirements" column) of all previous steps on which the current step depends. Instantiation steps explicitly list the instantiations used.

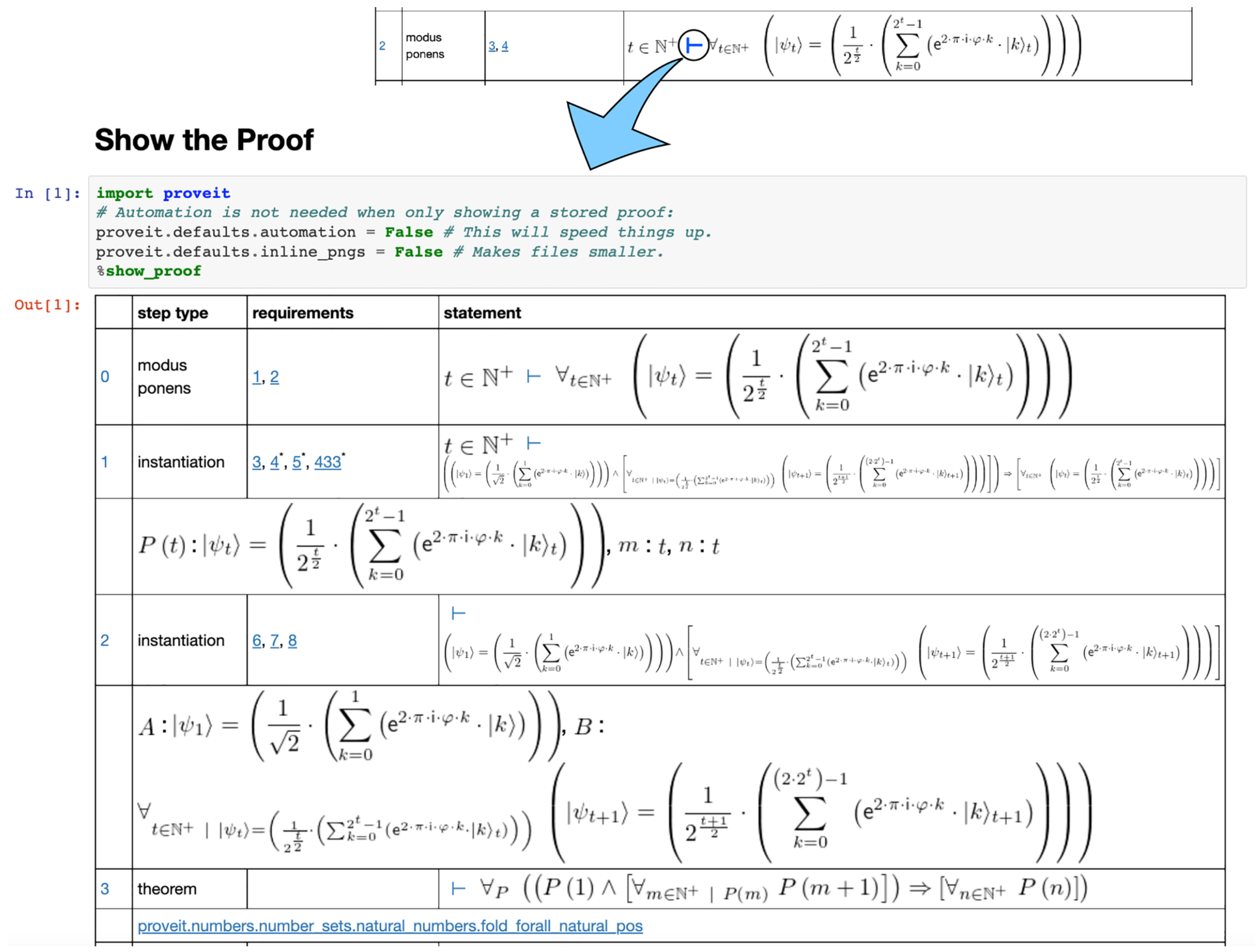
These proof characteristics make a formal proof in PROVE-IT particularly human readable and human checkable.

From a theorems page of a theory package in the PROVE-IT website [33], clicking on a theorem name brings up its proof notebook. One can then scroll down to the final cell with the %qed command. The output of this cell is the formal proof. A formal proof produced by PROVE-IT is often quite lengthy. Thus the formal proof steps appear in a nontraditional reverse (or top-down) order, which makes it easier for the user to see the main steps in a proof, only working down the tree to earlier or higher-numbered steps when desired. As an example, a snapshot of the first few lines of PROVE-IT's formal proof of the _psi_t_formula theorem (16) appears in Fig. 6. The steps are numbered along the left-hand side, with step 0 being the final theorem statement (and the root of the proof tree structure of derivations and axioms), with higher-numbered steps representing steps further down into the proof tree and supporting the lower-numbered steps at the top.

When referring to the order of the proof steps we will use the reverse order unless otherwise stated, i.e., higher-numbered steps are earlier steps.

Each step explicitly names the type of derivation step being taken and the previous (higher-numbered) steps required for the step. In Fig. 6, for example, step 2 applies *modus ponens* using the results from step 3 (a logical implication) and step 4 (the antecedent of the logical implication). Given the explicit listing of requirements for each step, one *could* manually track through the entire proof to see all the steps that contributed to the conclusion in step 2, but the system can actually do all the work for the user: Simply clicking on the blue turnstile symbol ⊢ in the statement of interest will bring up a new page (a webpage on the system's website or a new JUPYTER notebook on a developer's local computer) showing all the detailed steps leading to that particular conclusion, as illustrated in Fig. 7 for the subproof produced for step 2.

For long formal proofs, hand verification is tedious and unreliable, of course, but provides some assurance. PROVE-IT itself ensures that each derivation step is valid and that there are no circular dependencies, but PROVE-IT is a large and complicated piece of software that should not be blindly trusted. The development of a lightweight independent checker that is also open source with detailed documentation to garner more trust is left for future work. We aim to make this checker simple enough and with enough detail that other researchers could develop their own and further their confidence in the system and resulting proof certificates. This will demonstrate that PROVE-IT satisfies the de Bruijn criterion [46,47].



2	modus ponens	3, 4	$t \in \mathbb{N}^+ \vdash \forall_{t \in \mathbb{N}^+} \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)\right)$

Show the Proof

```
In [1]: import proveit
        # Automation is not needed when only showing a stored proof:
        proveit.defaults.automation = False # This will speed things up.
        proveit.defaults.inline_pngs = False # Makes files smaller.
        %show_proof
```

Out[1]:

	step type	requirements	statement
0	modus ponens	1, 2	$t \in \mathbb{N}^+ \vdash \forall_{t \in \mathbb{N}^+} \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)\right)$
1	instantiation	3, 4*, 5*, 433*	$t \in \mathbb{N}^+ \vdash \left(\left(\lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle\right)\right)\right)\right) \wedge \left[\forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}}\cdot\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\cdot\lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)\right]\right) \Rightarrow \left[\forall_{t \in \mathbb{N}^+} \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right)\right)\right]$
	$P(t) : \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_t\right)\right)\right), m : t, n : t$		
2	instantiation	6, 7, 8	$\vdash \left(\lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle\right)\right)\right)\right) \wedge \left[\forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}}\cdot\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\cdot\lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)\right]$
	$A : \lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle\right)\right)\right), B : \forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}}\cdot\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\cdot\lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)$		
3	theorem		$\vdash \forall_P \left(\left(P(1) \wedge \left[\forall_{m \in \mathbb{N}^+ \mid P(m)} P(m+1)\right]\right) \Rightarrow \left[\forall_{n \in \mathbb{N}^+} P(n)\right]\right)$
	proveit.numbers.number_sets.natural_numbers.fold_forall_natural_pos		

FIG. 7. Obtaining a subproof from within a proof in PROVE-IT: a snapshot of the first few lines of PROVE-IT's formal proof of step 2 in Fig. 6, obtained by clicking on the turnstile symbol in that line's judgment (shown circled at the top).

B. Theorem dependencies

Having a proof of a given statement means that we have produced a sequence of small steps that derive the desired statement from more fundamental and accepted statements. In our case, we derive statements that are specific to the quantum phase estimation algorithm from statements that are more fundamental facts in logic and set theory, number theory, trigonometry, linear algebra, statistics, and quantum mechanics. The complete list of the dependencies for each proven statement can be viewed from our website [33] by navigating to the theorems page of the `proveit.physics.quantum.QPE` package, clicking on the corresponding theorem name, and then clicking on the "dependencies" link.

Statements (axioms and theorems) are organized into theory packages in our system. Our universal QPE proofs derive from a number of statements from various packages. Table I lists the number of statements from each theory package (along with some examples of such statements) that, at the time of this writing, were logically required to derive the formal proofs. The majority of these statements are theorems that have not yet been proven in our system. Each of these is marked as a conjecture in the formal proof steps. It is a convenient feature in our system that conjectures may be posited and used (sometimes through automation) before they are proven and then noted as unproven in the lists of dependencies.[7] As these conjectures are eventually proven, the number of dependencies will decrease, eventually leaving a much smaller number of axiom dependencies.

Moreover, many axioms that currently appear as dependencies are merely label definitions and thus each serves as a (conservative) extension by definition (see [42], pp. 102–103, and [35], pp. 57–61). A future update of PROVE-IT should distinguish conservative definitions from axioms. After proving the unique existence of the label as it is defined, the definition will no longer be a dependency when used to prove a theorem unless that particular label appears in that particular theorem. Often in mathematics, convenient notation is used incidentally in the process of constructing a proof, but the validity of the proof is independent of this extraneous notational convenience. We want the dependencies to properly reflect this fact.

[7]This convenience requires no further action on the user's part, but is analogous to the explicit use of the `Accepted` command in COQ.

TABLE I. Combined number of axioms and (unproven) theorems in each theory package (along with some examples) upon which the three main QPE theorems, `qpe_exact` (31), `qpe_best_guarantee` (38), and `qpe_precision_guarantee` (53), logically depend at the time of this writing.

Theory package	No. of dependent axioms, theorems	Examples
`core_expr_types`	9, 22	$\lvert()\rvert = \lvert(1, 2, \ldots, 0)\rvert$ $\forall_{n\in\mathbb{N}}\{\forall_{x_1,\ldots,x_n,y_1,\ldots,y_n\lvert(x_1,\ldots,x_n)=(y_1,\ldots,y_n)}[f(x_1,\ldots,x_n) = f(y_1,\ldots,y_n)]\}$
`linear_algebra`	4, 23	$\forall_K\{\forall_{V\in_c \text{VecSpaces}(K)}[\forall_{x,y\in V}(x + y \in V)]\}$ $\forall_K(\forall_{V\in_c \text{VecSpaces}(K)}\{\forall_{a\in K}[\forall_{x\in V}(ax \in V)]\})$
`logic.booleans`	21, 9	$\forall_{m\in\mathbb{N}^+}[\forall_{A_1,\ldots,A_m\lvert A_1,\ldots,A_m}(A_1 \wedge \ldots \wedge A_m)]$ $\forall_{m\in\mathbb{N}^+}[\forall_{A_1,\ldots,A_m\lvert A_1,\ldots,A_m}(A_1 \vee \ldots \vee A_m)]$
`logic.equality`	6, 1	$\forall_x(x = x)$ $\forall_{x,y}[(y = x) = (x = y)]$ $\forall_{x,y,z\lvert x=y,y=z}(x = z)$ $\forall_{f,x,y\lvert x=y}[f(x) = f(y)]$ $\forall_{a,b,c,d\lvert a=b,b=c,c=d}(a = d)$
`logic.sets`	5, 14	$\forall_{A,B}\{(A \subseteq B) = [\forall_{x\in A}(x \in B)]\}$ $\forall_{A,B}[(A \subset B) = (A \subseteq B \wedge A \ncong B)]$ $\forall_{f,a}(\{\forall_{x,y\in A\lvert x\neq y}[f(x) \neq f(y)]\} \Rightarrow \{f \in [A \xrightarrow[\text{onto}]{\text{1-to-1}} f^{\rightarrow}(A)]\})$
`numbers.addition`	0, 44	$\forall_{A,B\in\mathbb{C}}(A + B \in \mathbb{C})$ $\forall_{A,B\in\mathbb{C}}(A + B = B + A)$ $\forall_{a\in\mathbb{C}}(a + 0 = a)$ $\forall_{a,x,y\in\mathbb{R}\lvert x<y}[(x + a) < (y + a)]$ $\forall_{a,b\lvert a<b}(a - b < 0)$ $\forall_{a,b\lvert a\neq b}(a - b \neq 0)$ $\forall_{a,b,c\in\mathbb{C}\lvert(a+b)=c}(c - b = a)$
`numbers.division`	0, 23	$\forall_{a,b\in\mathbb{C}\lvert b\neq0}(\frac{a}{b} \in \mathbb{C})$ $\forall_{x\in\mathbb{C}\lvert x\neq0}(\frac{x}{x} = 1)$ $\forall_{x\in\mathbb{C}\lvert x\neq0}(\frac{0}{x} = 0)$ $\forall_{x,y\in\mathbb{X}\lvert y\neq0}(\frac{-x}{y} = -\frac{x}{y})$ $\forall_{a,x,y\in\mathbb{R}^+\lvert x<y}(\frac{a}{x} > \frac{a}{y})$
`numbers.exponentiation`	0, 41	$\forall_{a\in\mathbb{R}^+,b,c\in\mathbb{C}}[(a^b)^c = a^{bc}]$ $\forall_{x\in\mathbb{C}}(x^1 = x)$ $\forall_{a,x,y\in\mathbb{R}\lvert x=y}(x^a = y^a)$ $\forall_{x\in\mathbb{C}}\{\forall_{n\in\mathbb{N}}[x^{2n} = (-x)^{2n}]\}$ $\forall_{a,b\in\mathbb{C}}(a^b \in \mathbb{C})$ $\forall_{a,x,y\in\mathbb{R}^+\lvert a>1,x<y}(a^x < a^y)$
`numbers.multiplication`	0, 33	$\forall_{a,b\in\mathbb{C}}(ab = ba)$ $\forall_{x\in\mathbb{C}}(1x = x)$ $\forall_{a,b\in\mathbb{C}}(ab \in \mathbb{C})$ $\forall_{a,b\in\mathbb{C}^{\neq0}}(ab \in \mathbb{C}^{\neq0})$ $\forall_{x,y\in\mathbb{C}}[(-x)y = -(xy)]$ $\forall_{a,x,y\in\mathbb{R}\lvert x<y,a<0}(ax > ay)$
`numbers.negation`	0, 16	$\forall_{a\in\mathbb{C}}[(-a) \in \mathbb{C}]$ $\forall_{a,b\in\mathbb{C}}[-(a + b) = (-a - b)]$ $\forall_{x\in\mathbb{C}}[-(-x) = x]$ $\forall_{x\in\mathbb{C}}[(-1)x = (-x)]$ $\forall_{x,y\in\mathbb{R}\lvert x<y}[(-x) > (-y)]$ $-0 = 0$
`numbers.number_sets`	2, 83	$\forall_{n\in\mathbb{N}}[(n + 1) \in \mathbb{N}]$ $0 \in \mathbb{N}$ $0 \in \mathbb{C}$ $i \in \mathbb{C}$ $\mathbb{R} \subset \mathbb{C}$ $\mathbb{Z} \subset \mathbb{C}$ $\forall_{a,b\in\mathbb{Z}}[\{a, \ldots, b\} \subset \mathbb{Z}]$ $\forall_{a,b\in\mathbb{Z}}[\forall_{n\in\{a,\ldots,b\}}(n \leqslant b)]$ $\mathbb{N} \subset \mathbb{Z}$
`numbers.numerals`	2, 36	$1 = 0 + 1$ $2 = 1 + 1$ $0 < 1$ $1 < 2$ $2 \times 2 = 4$ $3 \in \mathbb{N}$ $1 \in \mathbb{N}^+$ $2 \in \mathbb{N}^+$ $3 \in \mathbb{N}^+$ $\forall_{a,b,c}[\lvert(a, b, c)\rvert = \lvert(1, 2, \ldots, 3)\rvert]$
`numbers.ordering`, `numbers.rounding`, & `numbers.summation`	7, 29	$\forall_{x,y,z\lvert x<y,y<z}[x < z]$ $\forall_{x\in\mathbb{R}}(\lfloor x\rfloor \leqslant x)$ $\forall_{x\in\mathbb{R}}[\texttt{round}(x) \in \mathbb{Z}]$ $\forall_{x\in\mathbb{R}}[(x - \lfloor x\rfloor) < 1]$ $\forall_{x\in\mathbb{R}}[\texttt{round}(x) = \lfloor x + \frac{1}{2}\rfloor]$ $\forall_f[\forall_{a,b,c\in\mathbb{Z}}(\sum_{x=a}^{b} f(x) = \sum_{x=a+c}^{b+c} f(x - c))]$
misc. other `numbers` subpackages	0, 29	$\forall_{a,b\in\mathbb{C}}(\lvert a - b\rvert = \lvert b - a\rvert)$ $\forall_{\theta\in\mathbb{R}}(\lvert e^{i\theta}\rvert = 1)$ $[x \mapsto \frac{1}{x^2}] \in \texttt{MonDecFuncs}(\mathbb{R}^+)$ $\forall_{a,b\in\mathbb{R}^+\lvert a\leqslant b}[(\int_a^b \frac{1}{\ell^2} d\ell) \leqslant \frac{1}{a}]$
`physics.quantum`	3, 44	$\lvert0\rangle \in \mathbb{C}^2$ $\lvert1\rangle \in \mathbb{C}^2$ $\lVert\lvert0\rangle\rVert = 1$ $\lVert\lvert1\rangle\rVert = 1$ $\langle0\lvert1\rangle = 0$ $\forall_{n\in\mathbb{N}^+}[\text{FT}_n^{\dagger} \in U(2^n)]$ $\forall_{n\in\mathbb{N}^+}(C^n \in_c \texttt{HilbertSpaces})$
`statistics`	1, 7	$\forall_{\Omega\in\text{SampleSpaces}}(\forall_{X\in\Omega}\{\Pr(X) \in [0, 1]\})$
`trigonometry`	0, 5	$\forall_{\theta\in\mathbb{R}^+}[\sin(\theta) < \theta]$ $\forall_{a,b\in\mathbb{R}}[\lvert e^{ia} - e^{ib}\rvert = 2\sin(\frac{\lvert a-b\rvert}{2})]$

Circular dependencies are prevented, on both the local level of a formal proof and the broader level of theorem dependencies.

C. Lengths of the QPE proofs

As mentioned repeatedly, all of our proofs are viewable on the PROVE-IT website [33]. All of the theorems in the QPE theory package (both universal and local) are listed in Tables II and III, along with the number of user-invoked derivation commands utilized in each JUPYTER notebook and the number of steps in the eventual formal proof produced by PROVE-IT for that theorem. Table II includes the three main QPE theorems (indicated by an asterisk) and the major supporting theorems, ordered to parallel the presentation in Sec. II. Table III lists more minor theorems. Figure 8 presents a graphical representation of the interdependencies among the main and supporting theorems found in Table II.

The number of steps in a formal proof may be surprising. Many details are required in a formal proof. It is also likely

TABLE II. Number of manually invoked derivation commands (e.g., the calling of specific methods or the instantiation of a theorem to derive a new judgment) in each proof notebook (nb), and the number of steps (nodes in the formal proof tree) appearing in the corresponding formal proof for the three main QPE theorems (indicated by an asterisk) and major supporting theorems in the QPE theory package. The ordering of the theorems parallels the overview in Sec. II with theorems depending only on others higher in the list.

QPE theorem	No. of nb derivation commands	No. of steps in formal proof
`_psi_t_output` (12)	11	752
`_Psi_output` (19)	10	411
`_sample_space_bijection` (22)	4	273
`_outcome_prob` (23)	6	392
`_Omega_is_sample_space` (25) $\Omega \underset{c}{\in} \texttt{SampleSpaces}$	7	29
`_psi_t_formula` (16) $\forall_{t\in\mathbb{N}^+}\{\lvert\psi_t\rangle = [\frac{1}{2^{t/2}}(\sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}\lvert k\rangle_t)]\}$	24	685
`_alpha_m_evaluation` (27) $\forall_{m\in\{0,\ldots,2^t-1\}}(\alpha_m = \frac{1}{2^t}\sum_{k=0}^{2^t-1}(e^{-2\pi ikm/2^t}e^{2\pi i\varphi k}))$	8	404
`_alpha_ideal_case` (30) $(2^t\varphi \in \{0,\ldots,2^t-1\}) \Rightarrow (\alpha_{2^t\varphi} = 1)$	2	195
`qpe_exact` (31)*	15	161
`_alpha_m_mod_evaluation` (32) $\forall_{m\in\mathbb{Z}}(\alpha_{m \bmod 2^t} = \frac{1}{2^t}\sum_{k=0}^{2^t-1}(e^{-2\pi ikm/2^t}e^{2\pi i\varphi k}))$	12	161
`_alpha_m_mod_as_geometric_sum` (33) $\forall_{m\in\mathbb{Z}}(\alpha_{m \bmod 2^t} = \frac{1}{2^t}\sum_{k=0}^{2^t-1}(e^{2\pi i(\varphi-m/2^t)})^k)$	8	189
`_best_guarantee_delta_nonzero` (36) $(\delta_{b_r} \neq 0) \Rightarrow (\lvert\alpha_{b_r \bmod 2^t}\rvert^2 > \frac{4}{\pi^2})$	23	625
`_best_guarantee` (37) $\lvert\alpha_{\texttt{round}(2^t\varphi) \bmod 2^t}\rvert^2 > \frac{4}{\pi^2}$	24	259
`qpe_best_guarantee` (38)*	4	19
`_alpha_summed` (40) $\forall_{\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\mid\ell\neq 0}(\alpha_{b_f\oplus\ell} = \frac{1}{2^t}\frac{1-\exp[2\pi i(2^t\delta_{b_f}-\ell)]}{1-\exp[2\pi i(\delta_{b_f}-\ell/2^t)]})$	12	417
`_fail_sum_prob_conds_equiv_lemma` (46)	12	412
`_fail_sum` (45) $\forall_{e\in\{1,\ldots,2^{t-1}-2\}}\ ([P_{\text{fail}}](e) = \sum_{\ell=-2^{t-1}+1}^{-(e+1)}\lvert\alpha_{b_f\oplus\ell}\rvert^2 + \sum_{\ell=e+1}^{2^{t-1}}\lvert\alpha_{b_f\oplus\ell}\rvert^2)$	24	453
`_alpha_sqrd_upper_bound` (47) $\forall_{\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\mid\ell\neq 0}(\lvert\alpha_{b_f\oplus\ell}\rvert^2 \leqslant \frac{1}{4(\ell-2^t\delta_{b_f})^2})$	25	706
`_failure_upper_bound_lemma` (48)	16	303
`_failure_upper_bound` (49) $\forall_{e\in\{1,\ldots,2^{t-1}-2\}}([P_{\text{fail}}](e) \leqslant \frac{1}{2e} + \frac{1}{4e^2})$	28	547
`_success_complements_failure` (50) $\forall_{e\in\{1,\ldots,2^{t-1}-2\}}\{[P_{\text{success}}](e) = 1 - [P_{\text{fail}}](e)\}$	5	70
`_precision_guarantee_lemma_01` $(1 - \frac{1}{2(2^{t-n}-1)} - \frac{1}{4(2^{t-n}-1)^2}) > (1-\epsilon)$	27	589
`_precision_guarantee_lemma_02` $\forall_{m\in\{0,\ldots,2^t-1\}\mid\lvert m-b_f\rvert_{\text{mod } 2^t}\leqslant(2^{t-n}-1)}(\lvert\frac{m}{2^t}-\varphi\rvert_{\text{mod } 1} \leqslant 2^{-n})$	20	282
`_precision_guarantee` (52)	9	139
`qpe_precision_guarantee` (53)*	2	2

possible to substantially reduce the number of formal proof steps by making improvements in the implementation of the automation features that will be described in Sec. IV. We have focused our efforts on the capabilities of the automation features (to be discussed in Sec. IV) to minimize the number of derivation commands that are needed to generate the proof. Although the number of formal proof steps listed in Table II is only a rough estimate of some ideal number of steps, it is interesting to note how it compares with the number of derivation commands. A given derivation command can generate several formal proof steps in order to fill the gaps that result from skipping obvious steps in the sequence of derivation commands.

D. Expression structures and LATEX formatting

Expressions in PROVE-IT have an internal directed acyclic graph (DAG) structure that is critically important to its interpretation and the allowable proof steps. The LATEX formatting makes expressions much more human readable, but it is important to note that this formatting is not guaranteed to completely disambiguate its internal structure. In fact, there is really no safeguard in place to ensure that the formatting is true to its structure. Therefore, for full end-to-end verification, in addition to checking the derivation steps and theorem dependencies, one should also check the structure of the final derived statement to ensure it is indeed the statement that was intended to be proven and one should check the structures of the statements that were logically required (e.g., the axioms and conjectures).

Details about PROVE-IT's core expression types and structure, as well as their relation with derivation steps, are discussed in Ref. [32]. PROVE-IT has been developed particularly with quantum algorithms and quantum circuits in mind. Quantum circuit expressions in this work are good examples of the distinction between LATEX formatting and

TABLE III. Number of manually invoked derivation commands in the proof notebook and the number of steps appearing in the formal proof, for each of the more minor theorems in the QPE theory package. These minor theorems typically involve simple algebraic relationships, closure properties, and domain specifications.

QPE theorem	No. of derivative nb commands	No. of steps in formal proof
`_two_pow_t_is_nat_pos` $2^t \in \mathbb{N}^+$	1	7
`_two_pow_t_minus_one_is_nat_pos` $2^{t-1} \in \mathbb{N}^+$	1	20
`_psi_t_ket_is_normalized_vec` $\forall_{t\in\mathbb{N}^+}(\lvert\psi_t\rangle \in \mathbb{C}^{2^t} \wedge \lVert\lvert\psi_t\rangle\rVert = 1)$	4	329
`_Psi_ket_is_normalized_vec` $\lvert\Psi\rangle \in \mathbb{C}^{2^t} \wedge \lVert\lvert\Psi\rangle\rVert = 1$	4	27
`_best_floor_is_int` $b_f \in \mathbb{Z}$	2	16
`_best_floor_is_in_m_domain` $b_f \in \{0, \ldots, 2^t - 1\}$	8	56
`_best_round_is_int` $b_r \in \mathbb{Z}$	2	16
`_e_value_ge_two` $(2^{t-n} - 1) \geqslant 2$	8	249
`_e_value_in_e_domain` $(2^{t-n} - 1) \in \{1, \ldots, 2^{t-1} - 2\}$	11	317
`_mod_add_closure` $\forall_{a,b\in\mathbb{Z}}(a \oplus b \in \{0, \ldots, 2^t - 1\})$	3	12
`_phase_is_real` $\varphi \in \mathbb{R}$	1	12
`_delta_b_is_real` $\forall_{b\in\mathbb{Z}}(\delta_b \in \mathbb{R})$	3	25
`_scaled_delta_b_floor_in_interval` $2^t\delta_{b_f} \in [0, 1)$	9	108
`_scaled_delta_b_round_in_interval` $2^t\delta_{b_r} \in [-\frac{1}{2}, \frac{1}{2})$	15	167
`_delta_b_in_interval` $\forall_{b\in\{b_f,b_r\}}\{\delta_b \in (-\frac{1}{2}, \frac{1}{2}]\}$	11	324
`_alpha_are_complex` $\forall_{m\in\{0,\ldots,2^t-1\}}(\alpha_m \in \mathbb{C})$	2	20
`_delta_b_is_zero_or_non_int` $\forall_{b\in\{b_f,b_r\}}(\delta_b = 0 \vee \delta_b \notin \mathbb{Z})$	15	150
`_scaled_delta_b_is_zero_or_non_int` $\forall_{b\in\{b_f,b_r\}}(2^t\delta_b = 0 \vee 2^t\delta_b \notin \mathbb{Z})$	16	167
`_scaled_delta_b_not_eq_nonzeroInt` $\forall_{b\in\{b_f,b_r\},\ell\in\mathbb{Z}\mid\ell\neq0}(2^t\delta_b \neq \ell)$	9	31
`_delta_b_not_eq_scaledNonzeroInt` $\forall_{b\in\{b_f,b_r\},\ell\in\mathbb{Z}\mid\ell\neq0}(\delta_b \neq \frac{\ell}{2^t})$	6	56
`_delta_b_floor_diff_in_interval` $\forall_{l\in\{-2^{t-1}+1,\ldots,2^{t-1}\}}(\delta_{b_f} - \frac{\ell}{2^t} \in [-\frac{1}{2}, \frac{1}{2}))$	13	200
`_non_int_delta_b_diff` $\forall_{b\in\{b_f,b_r\},\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\mid\ell\neq0}(\delta_b - \frac{\ell}{2^t} \notin \mathbb{Z})$	14	233
`_scaled_abs_delta_b_floor_diff_interval` $\forall_{\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\mid\ell\neq0}(\pi\lvert\delta_{b_f} - \frac{\ell}{2^t}\rvert \in (0, \frac{\pi}{2}])$	9	93
`_pfail_in_real` $\forall_{e\in\{1,\ldots,2^{t-1}-2\}}([P_{\text{fail}}](e) \in \mathbb{R})$	6	44
`_phase_from_best_with_delta_b` $\forall_{b\in\mathbb{Z}}(\varphi = \frac{b}{2^t} + \delta_b)$	3	43
`_modabs_in_full_domain_simp` $\forall_{-2^{t-1}+1,\ldots,2^{t-1}}(\lvert\ell\rvert_{\text{mod } 2^t} = \lvert\ell\rvert)$	6	213

internal structures. PROVE-IT allows us to present quantum algorithms, as well as related axioms, definitions, and theorems in quantum circuit form (utilizing Q-circuit [48]), all of which provide a more intuitive and direct understanding for the user, but which then also have underlying representations in PROVE-IT's core in terms of nested tuples or nested arrays of related circuit details, which allow the manipulation of circuits and instantiation of circuit-based axioms and theorems (see Fig. 9 for an example).

IV. HOW WE CONSTRUCTED OUR PROOFS

Our primary objective is to make formal theorem proving as straightforward and natural as it is to develop and present a more informal proof. So it is vital that our system allows a user to skip obvious steps in a derivation and fill in details automatically. In this section we describe some of the system capabilities in this regard and provide examples from interactive proof steps in the QPE theory package. The system capabilities include (1) manual instantiation of axioms and theorems, (2) the availability of a wide variety of user-invoked derivation commands (tactics) that perform instantiations indirectly for a higher-level purpose (e.g., distributing, factoring, deducing new bounds, etc.), (3) goal-driven derivations that may be invoked directly or indirectly, (4) the use of canonical forms to recognize equivalent expressions and guide automated proof steps, (5) completely automatic incidental derivations performed whenever a judgment is established or assumptions are made, and (6) widespread user-configurable automatic simplification of expressions. As the examples will show, most proof steps involve a combination of these capabilities working in concert.

A. Manual instantiation

Any universally quantified statement (such as an explicitly imported axiom or theorem) may be instantiated with specific instances of the quantified variables. This can actually be quite powerful on its own, with an ability to expand ranges (e.g., $x_1y_1 + \cdots + x_ny_n$), automatically attempt to prove required conditions of the quantifiers as needed (see Sec. IV C), and take advantage of automatic simplification of the instantiated formula (see Sec. IV F), otherwise providing an informative error message (specifying, e.g., that some instantiation condition could not be proven).

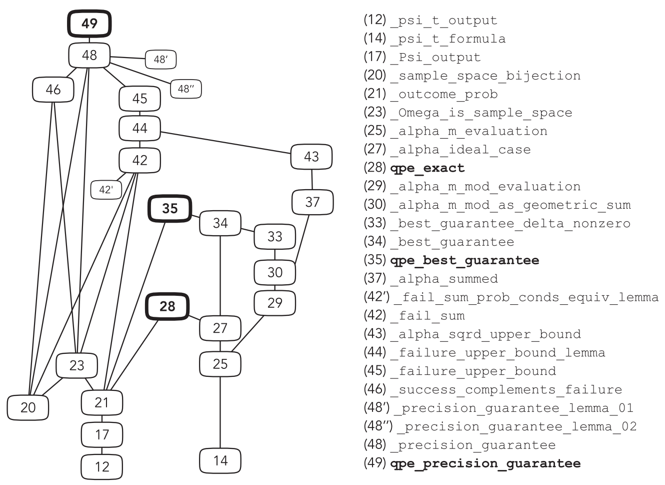

49
48
48′
48″
46
45
44
43
42
42′
37
35
34
33
30
29
28
27
25
23
21
20
17
14
12
(12) _psi_t_output
(14) _psi_t_formula
(17) _Psi_output
(20) _sample_space_bijection
(21) _outcome_prob
(23) _Omega_is_sample_space
(25) _alpha_m_evaluation
(27) _alpha_ideal_case
(28) qpe_exact
(29) _alpha_m_mod_evaluation
(30) _alpha_m_mod_as_geometric_sum
(33) _best_guarantee_delta_nonzero
(34) _best_guarantee
(35) qpe_best_guarantee
(37) _alpha_summed
(42′) _fail_sum_prob_conds_equiv_lemma
(42) _fail_sum
(43) _alpha_sqrd_upper_bound
(44) _failure_upper_bound_lemma
(45) _failure_upper_bound
(46) _success_complements_failure
(48′) _precision_guarantee_lemma_01
(48″) _precision_guarantee_lemma_02
(48) _precision_guarantee
(49) qpe_precision_guarantee

FIG. 8. Graphical depiction of the dependencies among the main theorems appearing in Table II. Numbering corresponds to equation numbers in the text (except for the primed numbers appearing in smaller boxes, for lemmas that do not explicitly appear in the text). The three main universal theorems appear in bold. An upward link means that the higher theorem explicitly depends on the lower theorem.

An example of manual (i.e., explicitly user-invoked) instantiation is shown in Fig. 10, excerpted from the proof notebook for `_alpha_summed` (40). The `_alpha_m_mod_as_geometric_sum` theorem shown in cell `[9]` is instantiated in cell `[10]` with quantified variable m instantiated as the expression $(b_f + \ell)$, which means that the instantiation process must be able to prove or verify $b_f + \ell \in \mathbb{Z}$. That required condition is automatically proven from $b_f \in \mathbb{Z}$ in cell `[8]` and the assumption $\ell \in \{-2^{t-1}+1, \ldots, 2^{t-1}\}$ (which is a default assumption set earlier in the notebook); this assumption then appears in the judgment produced by the instantiation in cell `[10]`.

Two more examples of manual instantiations are shown in Fig. 11, excerpted from the proof notebook for `_psi_t_formula` (16). Cells `[4]` and `[5]` show the general induction theorem for positive naturals and its instantiation with the specifics for the `_psi_t_formula` theorem (the variables m and n instantiated with t and the function P instantiated with the summation formula). Cells `[8]` and `[9]` show `_psi_t_def` (13) and its instantiation for $t = 1$, respectively.

B. User-invoked derivations (commands and tactics)

A user-invoked derivation typically utilizes an `Expression` class method that contains the instructions to determine axiom(s) or theorem(s) to invoke and appropriate instantiations of their quantified variables.[8] This frees the user from having to find the right axiom(s) or theorem(s), look up which variables need to be instantiated, and figure out exactly how to instantiate them. Some of these methods are more complicated and powerful, with provability checks, recursion, and/or postprocessing of resulting judgments.

Some examples of such user-invoked derivations are shown in Fig. 12, excerpted from the proof notebook for `_alpha_sqrd_upper_bound` (47). The `NumberOperation` class method `deduce_bound()`, employed in cells `[30]` and `[33]`, is a particularly powerful method. It propagates the effects of any number of bounds on subexpressions. In its internal workings, the `deduce_bound()` step in cell `[33]` also employs goal-driven derivations (see Sec. IV C) to satisfy requirements (e.g., establishing the signs of the numerator and denominator) as well as automatic simplification (see Sec. IV F) in canceling out a factor of 2.

C. Goal-driven derivations

A goal-driven derivation is an attempt to prove a particular statement indicated as the goal in advance. Goal-driven derivations are initiated either directly (manually) or indirectly (e.g., when an instantiation is made and conditions must be satisfied). A user may directly invoke the `.prove()` method on an expression, which instructs PROVE-IT to attempt to prove that expression, or it may be invoked indirectly through cascading requirements (e.g., the user wants to prove statement A but statement B must be proven as a prerequisite to deriving A).

Each `Expression` class has its own tactics for trying to prove something of its own type. As a deliberate design choice in PROVE-IT, we do *not* perform a deep, combinatorial search to find a tactic that works. Thus, simply calling `.prove()` on a formula of interest will often fail. As a guiding principle, we only invest in a single tactic for a given goal-driven derivation. If that fails, an exception is raised. To determine which one tactic to attempt, we perform efficient provability checks. The provability checks are neither exhaustive nor fail proof. We only check if something is readily (or obviously) provable. We may assume that expressions are properly typed (e.g., given $a + b$, assuming a and b are numbers) even though an actual proof will have to ensure the types are proper. These checks simply serve as a guide to choosing a tactic. The goal in PROVE-IT is to skip obvious steps, not provide fully automated theorem proving.

An indirect (automatic) goal-driven derivation was already encountered in the instantiation step of cell `[10]` in Fig. 10, with the system proving in its internal workings that $(b_f + \ell) \in \mathbb{Z}$ to satisfy the condition of the quantified statement being instantiated. An example of a directly (manually) invoked goal-driven derivation is shown in Fig. 13, excerpted from the proof notebook for `_precision_guarantee` (52). In this particular instance, the inequality is proven by automatically recognizing that a simple substitution can be made on the left-hand side of a previously known inequality. More generally, PROVE-IT may attempt a bidirectional search among known number-ordering relations ($<$, $\leqslant$, and $=$) to see if some inequality may be proven via transitivity (e.g., $a < d$ can be proven automatically if we know $a < b$, $b = c$, and $c \leqslant d$). The same bidirectional search algorithm can be utilized in goal-driven derivations of other transitive relations (e.g., subset relations $\subset$, $\subseteq$, and $\cong$).

[8]The object-oriented organization of the underlying code also then makes it easy to find desired methods.

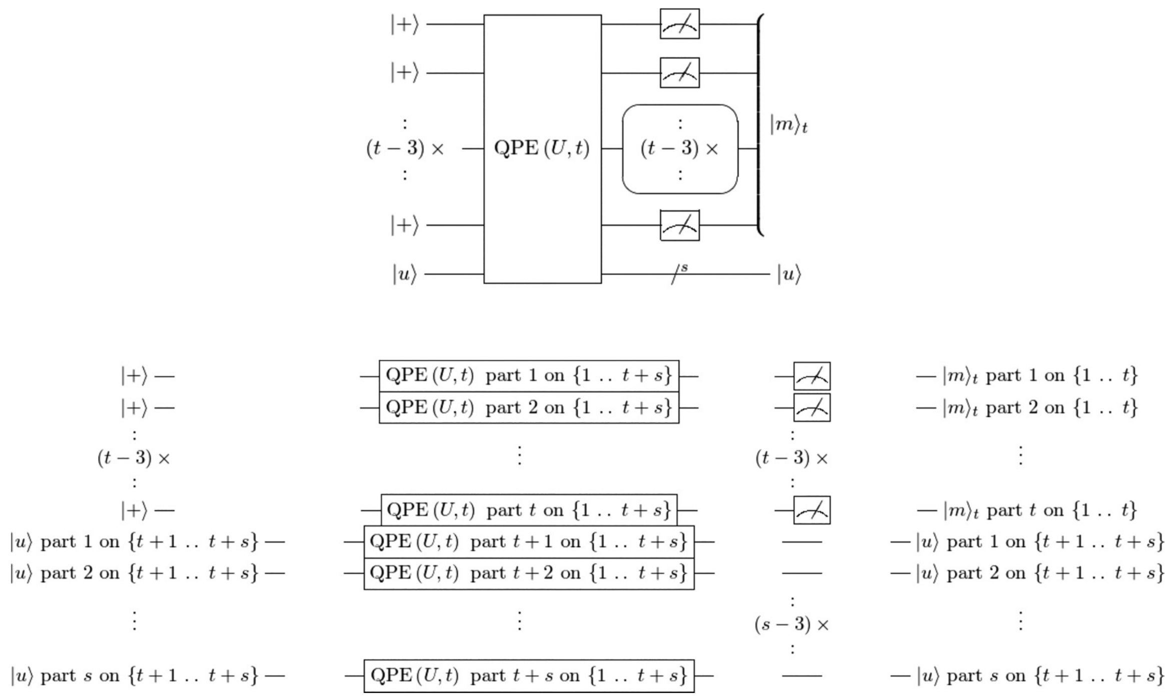

$|+\rangle$
$|+\rangle$
$(t-3)\times$
$|+\rangle$
$|u\rangle$
QPE (U,t)
$(t-3)\times$
$|m\rangle_t$
s
$|u\rangle$
QPE (U,t) part 1 on $\{1\,..\,t+s\}$
QPE (U,t) part 2 on $\{1\,..\,t+s\}$
QPE (U,t) part t on $\{1\,..\,t+s\}$
QPE (U,t) part $t+1$ on $\{1\,..\,t+s\}$
QPE (U,t) part $t+2$ on $\{1\,..\,t+s\}$
QPE (U,t) part $t+s$ on $\{1\,..\,t+s\}$
$|u\rangle$ part 1 on $\{t+1\,..\,t+s\}$
$|u\rangle$ part 2 on $\{t+1\,..\,t+s\}$
$|u\rangle$ part s on $\{t+1\,..\,t+s\}$
$(s-3)\times$
$|m\rangle_t$ part 1 on $\{1\,..\,t\}$
$|m\rangle_t$ part 2 on $\{1\,..\,t\}$
$|m\rangle_t$ part t on $\{1\,..\,t\}$

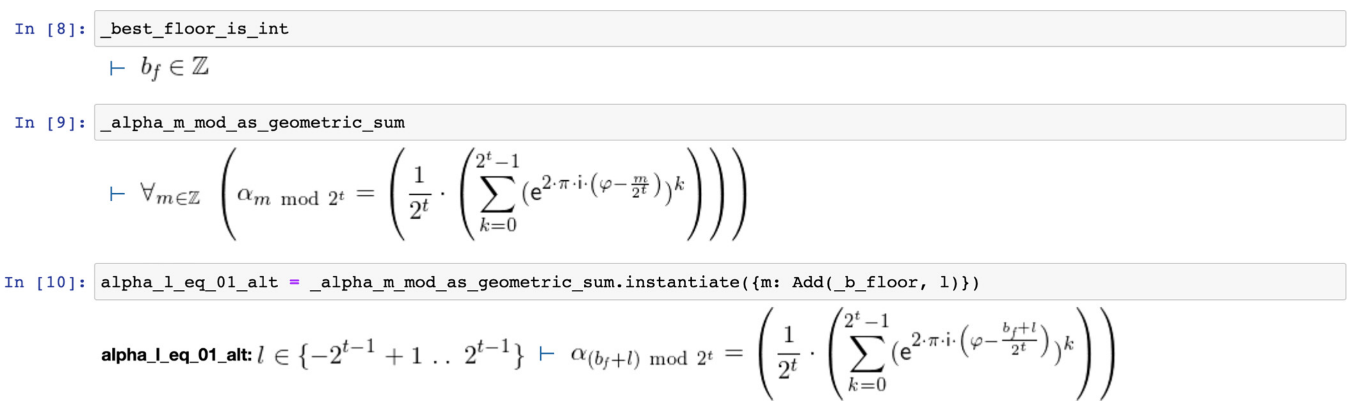
FIG. 9. Shown on top is an example circuit presentation in PROVE-IT and on bottom a stylized version of its underlying representation in the code illustrating its organization as a sequence of rows of circuit components nested within a sequence of columns. See Sec. III D for discussion.

D. Incidental derivations

Many simple derivations are performed automatically, without user direction, to proactively derive facts which may be useful for later derivations. When a judgment is proven or assumptions are made, incidental derivations are often made. These incidental derivations are directed by each `Expression` class.

Incidental derivations are not obvious to spot, since they do not appear explicitly in an interactive notebook step, but they assist derivations throughout. For example, proving that $A \vdash x \in \{2, \ldots, 10\}$ will incidentally derive (in the background, so to speak) $A \vdash x \geqslant 2$ and $A \vdash x \leqslant 10$ (where A represents any number of assumptions). When an equality judgment such as $A \vdash x = y$ is established, $A \vdash y = x$ is also automatically derived. The unfolding of a definition is a common incidental derivation as well, for example, $A \vdash \neg(x = y)$ is automatically derived upon proving that $A \vdash x \neq y$. Such incidental judgments are then available for deriving other conclusions, both automatic and user invoked. As a rule, PROVE-IT reserves automatic incidental derivations for obvious things and makes sure that incidental derivations can never cascade in a way that expands exponentially.

A pair of related examples of incidental derivation appear in cells `[19]` and `[20]` of Fig. 14, excerpted from the proof notebook for `_fail_sum` (45). In each example, assumptions generate incidental derivations that enable an instantiation and its simplification. We assume $e \in \{1, \ldots, 2^{t-1} - 2\}$ in both cases, which incidentally derives $e \geqslant 1$ and $e \leqslant 2^{t-1} - 2$.



```
In [8]: _best_floor_is_int
```

$$\vdash\ b_f \in \mathbb{Z}$$

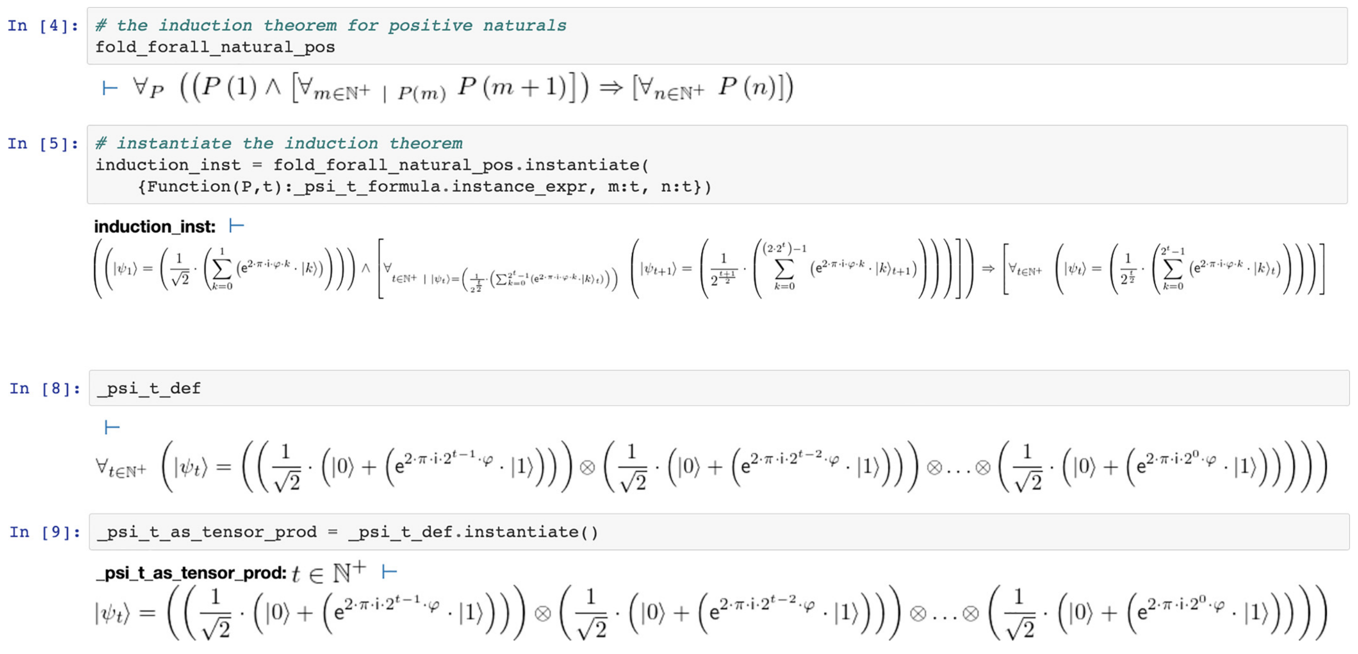
```
In [9]: _alpha_m_mod_as_geometric_sum
```

$$\vdash\ \forall_{m \in \mathbb{Z}} \left(\alpha_{m \bmod 2^t} = \left(\frac{1}{2^t} \cdot \left(\sum_{k=0}^{2^t - 1} \left(e^{2 \cdot \pi \cdot i \cdot \left(\varphi - \frac{m}{2^t}\right)}\right)^k\right)\right)\right)$$

```
In [10]: alpha_l_eq_01_alt = _alpha_m_mod_as_geometric_sum.instantiate({m: Add(_b_floor, l)})
```

alpha_l_eq_01_alt: $l \in \{-2^{t-1} + 1\,..\,2^{t-1}\} \vdash \alpha_{(b_f + l) \bmod 2^t} = \left(\frac{1}{2^t} \cdot \left(\sum_{k=0}^{2^t - 1} \left(e^{2 \cdot \pi \cdot i \cdot \left(\varphi - \frac{b_f + l}{2^t}\right)}\right)^k\right)\right)$

FIG. 10. Example of manual instantiation, excerpted from the proof notebook for `_alpha_summed` (16), instantiating `_alpha_m_mod_as_geometric_sum` (33) with $m = (b_f + \ell)$.



```
In [4]: # the induction theorem for positive naturals
        fold_forall_natural_pos
```

$$\vdash\ \forall_{P}\ \left(\left(P(1) \land \left[\forall_{m \in \mathbb{N}^{+}\ |\ P(m)}\ P(m+1)\right]\right) \Rightarrow \left[\forall_{n \in \mathbb{N}^{+}}\ P(n)\right]\right)$$

```
In [5]: # instantiate the induction theorem
        induction_inst = fold_forall_natural_pos.instantiate(
            {Function(P,t):_psi_t_formula.instance_expr, m:t, n:t})
```

induction_inst: $\vdash$

$$\left(\left(|\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(e^{2 \cdot \pi \cdot i \cdot \varphi \cdot k} \cdot |k\rangle\right)\right)\right)\right) \land \left[\forall_{t \in \mathbb{N}^{+}\ |\ |\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1}\left(e^{2 \cdot \pi \cdot i \cdot \varphi \cdot k} \cdot |k\rangle_t\right)\right)\right)}\ \left(|\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2 \cdot 2^t)-1} \left(e^{2 \cdot \pi \cdot i \cdot \varphi \cdot k} \cdot |k\rangle_{t+1}\right)\right)\right)\right)\right]\right) \Rightarrow \left[\forall_{t \in \mathbb{N}^{+}}\ \left(|\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(e^{2 \cdot \pi \cdot i \cdot \varphi \cdot k} \cdot |k\rangle_t\right)\right)\right)\right)\right]$$

```
In [8]: _psi_t_def
```

$\vdash$

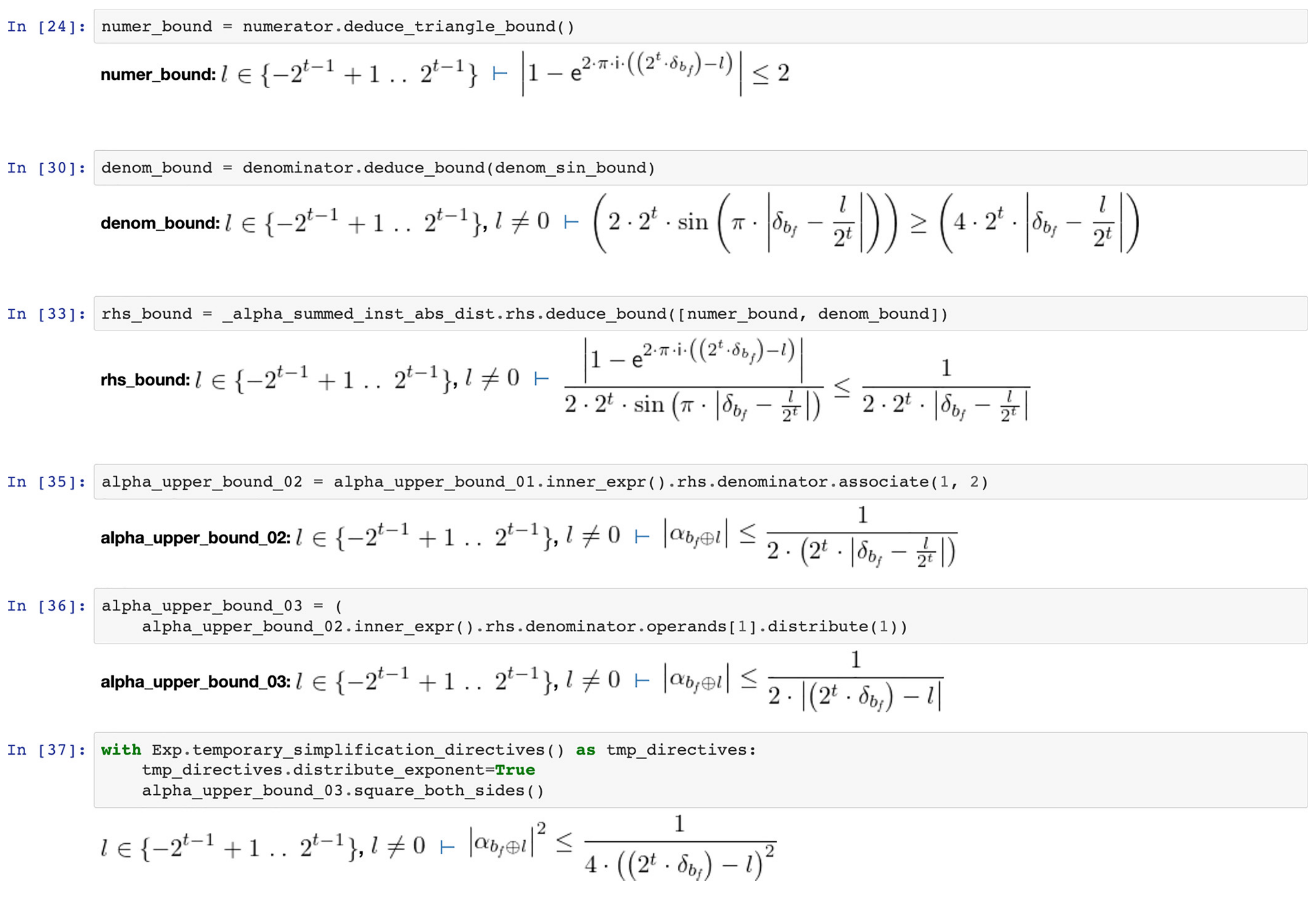
$$\forall_{t \in \mathbb{N}^{+}}\ \left(|\psi_t\rangle = \left(\left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2 \cdot \pi \cdot i \cdot 2^{t-1} \cdot \varphi} \cdot |1\rangle\right)\right)\right) \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2 \cdot \pi \cdot i \cdot 2^{t-2} \cdot \varphi} \cdot |1\rangle\right)\right)\right) \otimes \ldots \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2 \cdot \pi \cdot i \cdot 2^{0} \cdot \varphi} \cdot |1\rangle\right)\right)\right)\right)\right)$$

```
In [9]: _psi_t_as_tensor_prod = _psi_t_def.instantiate()
```

_psi_t_as_tensor_prod: $t \in \mathbb{N}^{+}\ \vdash$

$$|\psi_t\rangle = \left(\left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2 \cdot \pi \cdot i \cdot 2^{t-1} \cdot \varphi} \cdot |1\rangle\right)\right)\right) \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2 \cdot \pi \cdot i \cdot 2^{t-2} \cdot \varphi} \cdot |1\rangle\right)\right)\right) \otimes \ldots \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2 \cdot \pi \cdot i \cdot 2^{0} \cdot \varphi} \cdot |1\rangle\right)\right)\right)\right)$$

FIG. 11. Examples of manual instantiations, excerpted from the proof notebook for `_psi_t_formula` (16), showing the manual instantiation of the induction theorem for positive naturals (cells 4 and 5) and of the `_psi_t_def` (13) local QPE definition (cells 8 and 9).



```
In [24]: numer_bound = numerator.deduce_triangle_bound()
```

numer_bound: $l \in \{-2^{t-1}+1\ ..\ 2^{t-1}\}\ \vdash\ \left|1 - e^{2 \cdot \pi \cdot i \cdot \left(\left(2^t \cdot \delta_{b_f}\right) - l\right)}\right| \leq 2$

```
In [30]: denom_bound = denominator.deduce_bound(denom_sin_bound)
```

denom_bound: $l \in \{-2^{t-1}+1\ ..\ 2^{t-1}\}, l \neq 0\ \vdash\ \left(2 \cdot 2^t \cdot \sin\left(\pi \cdot \left|\delta_{b_f} - \frac{l}{2^t}\right|\right)\right) \geq \left(4 \cdot 2^t \cdot \left|\delta_{b_f} - \frac{l}{2^t}\right|\right)$

```
In [33]: rhs_bound = _alpha_summed_inst_abs_dist.rhs.deduce_bound([numer_bound, denom_bound])
```

rhs_bound: $l \in \{-2^{t-1}+1\ ..\ 2^{t-1}\}, l \neq 0\ \vdash\ \frac{\left|1 - e^{2 \cdot \pi \cdot i \cdot \left(\left(2^t \cdot \delta_{b_f}\right) - l\right)}\right|}{2 \cdot 2^t \cdot \sin\left(\pi \cdot \left|\delta_{b_f} - \frac{l}{2^t}\right|\right)} \leq \frac{1}{2 \cdot 2^t \cdot \left|\delta_{b_f} - \frac{l}{2^t}\right|}$

```
In [35]: alpha_upper_bound_02 = alpha_upper_bound_01.inner_expr().rhs.denominator.associate(1, 2)
```

alpha_upper_bound_02: $l \in \{-2^{t-1}+1\ ..\ 2^{t-1}\}, l \neq 0\ \vdash\ \left|\alpha_{b_f \oplus l}\right| \leq \frac{1}{2 \cdot \left(2^t \cdot \left|\delta_{b_f} - \frac{l}{2^t}\right|\right)}$

```
In [36]: alpha_upper_bound_03 = (
             alpha_upper_bound_02.inner_expr().rhs.denominator.operands[1].distribute(1))
```

alpha_upper_bound_03: $l \in \{-2^{t-1}+1\ ..\ 2^{t-1}\}, l \neq 0\ \vdash\ \left|\alpha_{b_f \oplus l}\right| \leq \frac{1}{2 \cdot \left|\left(2^t \cdot \delta_{b_f}\right) - l\right|}$

```
In [37]: with Exp.temporary_simplification_directives() as tmp_directives:
             tmp_directives.distribute_exponent=True
             alpha_upper_bound_03.square_both_sides()
```

$$l \in \{-2^{t-1}+1\ ..\ 2^{t-1}\}, l \neq 0\ \vdash\ \left|\alpha_{b_f \oplus l}\right|^2 \leq \frac{1}{4 \cdot \left(\left(2^t \cdot \delta_{b_f}\right) - l\right)^2}$$

FIG. 12. Examples of user-invoked derivation commands (tactics), excerpted from the proof notebook for `_alpha_sqrd_upper_bound` (47), showing user-invoked calls to `deduce_triangle_bound()`, `deduce_bound()`, `associate()`, `distribute()`, and `square_both_sides()` methods.

```
In [14]: greater(success_prob, _precision_guarantee_lemma_01.rhs).prove()
```

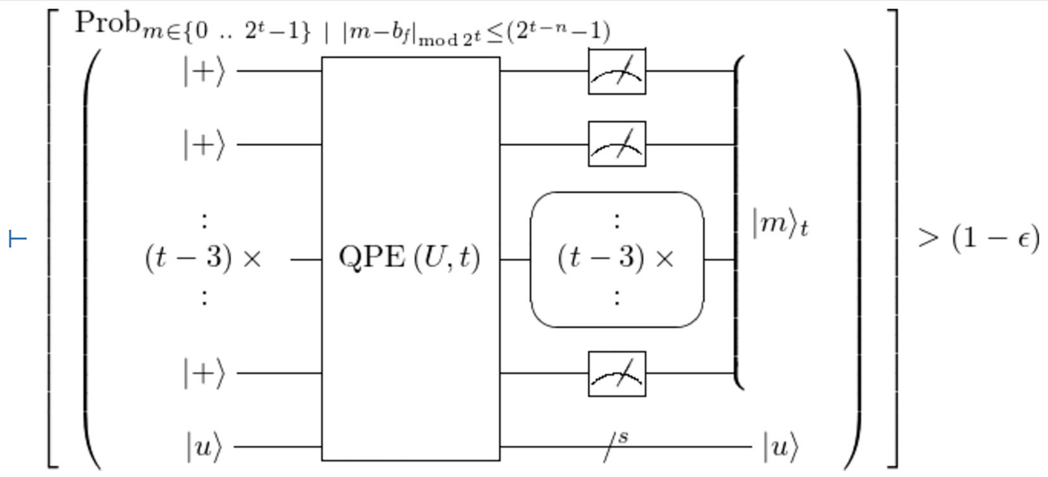


FIG. 13. Example of a goal-driven derivation, excerpted from the proof notebook for `_precision_guarantee` (52) showing the manual calling of `.prove()` on a formula of interest.

In cell `[19]`, we additionally assume $l \in \{e+1, \ldots, 2^{t-1}\}$, which incidentally derives $l \geqslant e+1$ and $l \leqslant 2^{t-1}$, from which it can automatically derive $l \in \{-2^{t-1}+1, \ldots, 2^{t-1}\}$ to satisfy the instantiation condition from `[18]`, and we can derive $l > 1$ so that $|l|$ simplifies to l (automatic simplification will be discussed in Sec. IV F). In cell `[20]` we additionally assume $l \in \{-2^{t-1}+1, \ldots, -(e+1)\}$, from which we can also derive $l \in \{-2^{t-1}+1, \ldots, 2^{t-1}\}$, but in this case $l < 1$ such that $|l|$ simplifies to $-l$. Incidental and goal-driven derivations are ubiquitous and often occur together like this in a bidirectional fashion and can be very powerful.

E. Equating expressions with the same canonical forms

In mathematics, there are often multiple ways of expressing the same thing with uninteresting or insignificant differences. In some instances, there is no difference in the inherent meaning and it is simply a style choice. For example, $\forall_{x_1,\ldots,x_n} P(x_1, \ldots, x_n)$ is no different than $\forall_{y_1,y_2,\ldots,y_n} P(y_1, y_2, \ldots, y_n)$ in its meaning (simply changing the internal x label to y and changing the presentation of the ranges). Another example is the total-ordering style of a conjunction of transitive relations; for example, $(a < b) \land (b < c)$ can be presented as $a < b < c$ in PROVE-IT using a style option. PROVE-IT offers flexibility in choosing the presentation style for such things while treating them as logically identical. In other cases, such as $2x/2 + 6$ vs $2 \times 3 + x$, the inherent meaning is different but it is straightforward to derive the equality. For the latter scenario, we use canonical forms. The choice of the canonical form (e.g., whether to use $6 + x$ or $x + 6$) has no bearing on proof generation or presentation and so we omit details here about how such canonical forms are generated. The important thing is that we choose consistent conventions to exploit knowledge about canonical form equivalence classes (sets of expressions that have the same canonical form), which then help drive proof automation.

Some simple examples of the use of canonical forms are shown in Fig. 15, excerpted from the proof notebook for `_best_guarantee_delta_nonzero` (36). The user-invoked method `deduce_linear_bound()` called in cell `[27]` needs to know that the angle $\theta = 2^t \pi |\delta_{b_r}| \leqslant \frac{\pi}{2}$, which the system recognizes through canonical forms as equivalent to the

```
In [18]: _modabs_in_full_domain_simp
```

$$\vdash \forall_{l \in \{-2^{t-1}+1\ ..\ 2^{t-1}\}} \left(|l|_{\mathrm{mod}\ 2^t} = |l|\right)$$

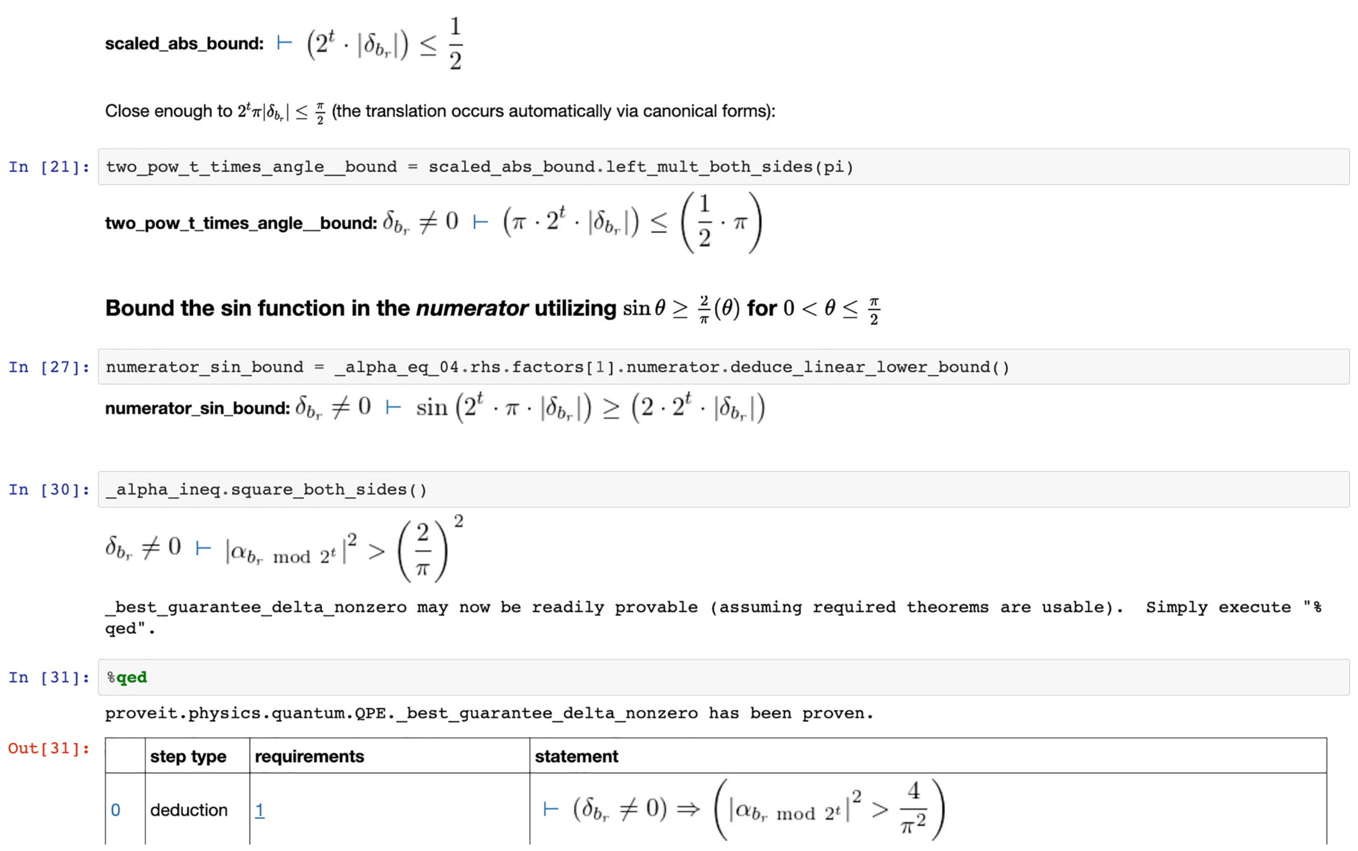
```
In [19]: absmod_posdomain_simp = _modabs_in_full_domain_simp.instantiate(
             assumptions=pos_domain_assumptions)
```

absmod_posdomain_simp: $e \in \{1\ ..\ 2^{t-1} - 2\}, l \in \{e+1\ ..\ 2^{t-1}\} \vdash |l|_{\mathrm{mod}\ 2^t} = l$

```
In [20]: absmod_negdomain_simp = _modabs_in_full_domain_simp.instantiate(
             assumptions=neg_domain_assumptions)
```

absmod_negdomain_simp: $e \in \{1\ ..\ 2^{t-1} - 2\}, l \in \{-2^{t-1}+1\ ..\ -(e+1)\} \vdash |l|_{\mathrm{mod}\ 2^t} = (-l)$

FIG. 14. Incidental derivation examples from an excerpt. of the proof notebook for `_fail_sum` (45).



scaled_abs_bound: $\vdash \left(2^t \cdot |\delta_{b_r}|\right) \leq \frac{1}{2}$

Close enough to $2^t\pi|\delta_{b_r}| \leq \frac{\pi}{2}$ (the translation occurs automatically via canonical forms):

```
In [21]: two_pow_t_times_angle__bound = scaled_abs_bound.left_mult_both_sides(pi)
```

two_pow_t_times_angle__bound: $\delta_{b_r} \neq 0 \vdash \left(\pi \cdot 2^t \cdot |\delta_{b_r}|\right) \leq \left(\frac{1}{2} \cdot \pi\right)$

Bound the sin function in the *numerator* utilizing $\sin\theta \geq \frac{2}{\pi}(\theta)$ **for** $0 < \theta \leq \frac{\pi}{2}$

```
In [27]: numerator_sin_bound = _alpha_eq_04.rhs.factors[1].numerator.deduce_linear_lower_bound()
```

numerator_sin_bound: $\delta_{b_r} \neq 0 \vdash \sin\left(2^t \cdot \pi \cdot |\delta_{b_r}|\right) \geq \left(2 \cdot 2^t \cdot |\delta_{b_r}|\right)$

```
In [30]: _alpha_ineq.square_both_sides()
```

$\delta_{b_r} \neq 0 \vdash |\alpha_{b_r \bmod 2^t}|^2 > \left(\frac{2}{\pi}\right)^2$

```
_best_guarantee_delta_nonzero may now be readily provable (assuming required theorems are usable).  Simply execute "%
qed".
```

```
In [31]: %qed
proveit.physics.quantum.QPE._best_guarantee_delta_nonzero has been proven.
```

Out[31]:

	step type	requirements	statement
0	deduction	1	$\vdash (\delta_{b_r} \neq 0) \Rightarrow \left(|\alpha_{b_r \bmod 2^t}|^2 > \frac{4}{\pi^2}\right)$

FIG. 15. Examples of equivalent canonical forms allowing automatic derivation steps, excerpted from the proof notebook for `_best_guarantee_delta_nonzero` (36).

inequality derived in cell `[21]`. Later, the interactive steps have established a formula in cell `[30]` involving the expression $(\frac{2}{\pi})^2$, while the theorem to be proven uses the expression $\frac{4}{\pi^2}$. While $(\frac{2}{\pi})^2$ and $\frac{4}{\pi^2}$ are distinct expression forms, they have the same canonical form. PROVE-IT determines that one can be readily transformed into the other, and thus no further manual steps are needed in the proof notebook.

F. Automatic simplifications

Why not automatically simplify everything to its canonical form? This is not always desirable and sometimes counterproductive. For example, $y + z/x$ may be the desired form even though $z/x + y$ may happen to be the canonical form. There are certain kinds of simplification that are sensible, but not always. It makes sense to simplify $y + 2x/2$ to $y + x$ (without changing the order in case it is preferable to the user for their purpose). It is not so obvious which of the forms $2x + 2y$ versus $2(x + y)$ should be preferred automatically, even though they have the same canonical form. PROVE-IT's solution is to allow fairly straightforward automatic simplifications that are configurable, that is, the user can manually disable autosimplifications or explicitly change default simplification directives for each expression type.

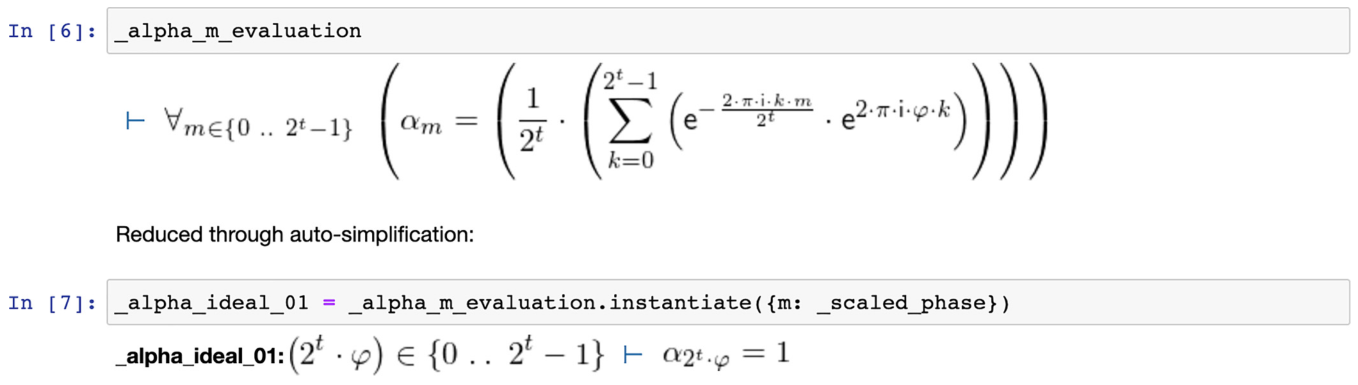
Automatic simplifications are performed as a postprocessing step for any derivation command unless it has been disabled. After implementing a theorem-instantiation step, for example, PROVE-IT traverses the resulting expression to simplify subexpressions according to the active simplification directives. Simplification substitutions are explicitly proven in the formal proof that is constructed.

A fairly dramatic example of such automatic simplification appears in Fig. 16, drawn from the proof notebook for `_alpha_ideal_case` (30). The expression in cell `[7]` resulting from the manual instantiation of the `_alpha_m_evaluation` theorem in `[6]` is greatly reduced through autosimplification. When m is instantiated as the `_scaled_phase` $2^t\varphi$, the product of exponentials in the summand is automatically reduced to 1 and then the summation of the constant summand 1 is automatically reduced and eventually canceled with the coefficient $\frac{1}{2^t}$. Such autosimplification saves many steps throughout our QPE derivations.

V. EXAMINING A SPECIFIC PROOF IN THE QPE PACKAGE

In previous sections we have provided scattered examples of both derivation commands fed into PROVE-IT in the process of proving QPE-specific theorems as well as some formal proof output. All of the proof constructions and formal output for QPE-specific theorems may be viewed on the PROVE-IT website [33]. There is too much material there to include in this article. However, to provide a sense for how the pieces come together, this section will focus on `_psi_t_formula` (16) as one particular example.

A subset of the `input/output` cells for the JUPYTER notebook used to construct the proof of `_psi_t_formula` is displayed in Figs. 17 and 18. While many steps are omitted for



```
In [6]: _alpha_m_evaluation
```

$$\vdash \forall_{m \in \{0 \,..\, 2^t - 1\}} \left(\alpha_m = \left(\frac{1}{2^t} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{-\frac{2 \cdot \pi \cdot \mathsf{i} \cdot k \cdot m}{2^t}} \cdot \mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \right) \right) \right) \right)$$

Reduced through auto-simplification:

```
In [7]: _alpha_ideal_01 = _alpha_m_evaluation.instantiate({m: _scaled_phase})
```

_alpha_ideal_01: $(2^t \cdot \varphi) \in \{0 \,..\, 2^t - 1\} \vdash \alpha_{2^t \cdot \varphi} = 1$

FIG. 16. Example of automatic simplification, drawn from the proof notebook for `_alpha_ideal_case` (30).

brevity, these chosen steps show the basic flow of this inductive proof and illustrate how derivation commands (discussed in Sec. IV) are used to move the proof along. The final step is to issue a `%qed` command to produce the formal proof, the first few lines of which are shown in Fig. 19. Note that the formal proof contains all of the fine details to make an airtight proof, each step of which is easy to confirm by human or machine (in principle). With 43 user-invoked `input/output` cells to construct the `_psi_t_formula` proof (which includes 24 distinct derivation commands), a formal proof of 684 steps is generated with all of the fine details to derive the `_psi_t_formula` from other proven theorems or more fundamental facts. We expect the formal proof sizes to reduce to some extent over time as PROVE-IT routines develop better strategies to generate succinct formal proofs, but the relatively large formal proof size does convey how much PROVE-IT is doing to fill in the obvious steps omitted in the proof construction.

The readability of both the proof construction `input/output` cells and the formal proof output offers interesting possibilities for communicating proofs and concepts, clearly and succinctly, among experts. In the Appendix we explore the potential connections between proofs in PROVE-IT and structured proofs intended for proof communication using `_psi_t_formula` as an example.

We can also compare 14 steps in the structured informal proof of the Appendix with the 24 derivation commands used to prove `_psi_t_formula`. This ratio is analogous to what de Bruijn called a loss factor [49] and Wiedijk has called the de Bruijn factor [50]. Wiedijk reported finding a de Bruijn factor of about 4 across a variety of texts and computer-proof systems. This comparison suggests a loss factor of roughly 2 for this particular example (though this ratio really depends upon the level of detail provided in the informal proof).

Finally, as discussed in Sec. III B, the user has easy access to a complete summary list of all conjectures, axioms, and conservative definitions required in the formal proof (as well as a list of all theorems that depend on the theorem in question) through its dependencies page. An excerpt of the dependencies page for `_psi_t_formula` (16) is shown in Fig. 20.

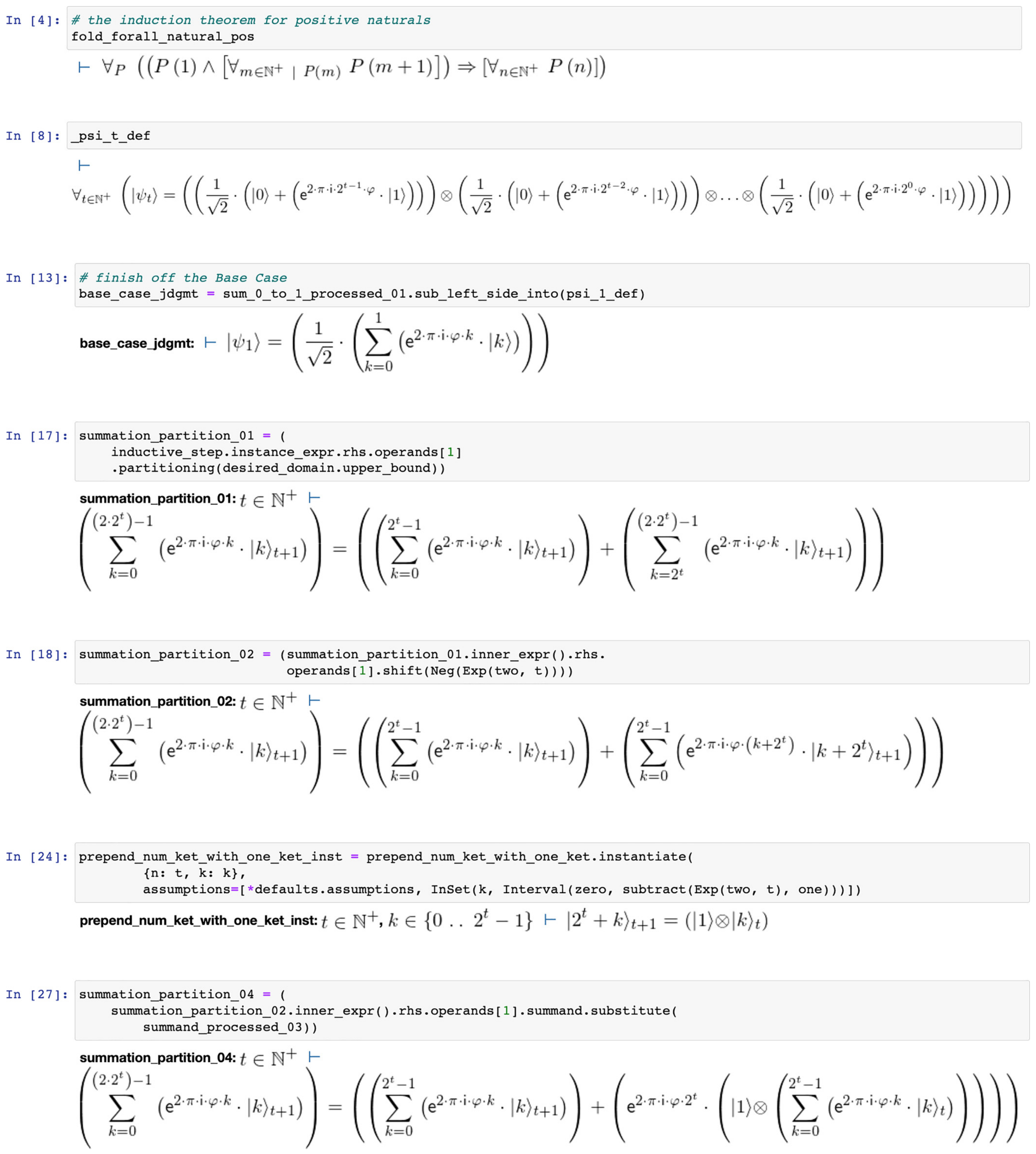
VI. COMPARISONS WITH OTHER APPROACHES

PROVE-IT is designed to enable robust theorem proving via a user-interactive process that can mimic informal proof styles. It is not intended as a black-box automated theorem-proving system. The automation in PROVE-IT is mainly intended as a means to skip obvious steps that would normally be skipped in an informal proof. Furthermore, PROVE-IT generates human-readable formal proofs; they can be quite long, but any given derivation step can be understood with a small amount of training. To our knowledge, PROVE-IT is unique in these respects.

We are not the first to provide a formal verification of QPE-related claims; related recent work on this problem can be found in works by Liu *et al.* [28], Hietala *et al.* [40], Chareton *et al.* [24], and Bauer-Marquart *et al.* [51]. Our work is unique in providing a verification for Nielsen and Chuang's [31] QPE-related precision guarantee `qpe_precision_guarantee` (53) and in providing not just verifications but detailed formal human-readable proofs of the exact, general, and precision-guarantee cases. This aspect of our proof attempt is elaborated further below.

Liu *et al.* [26] implemented quantum Hoare logic (QHL) [29] in ISABEL/HOL and PYTHON and reported having the first mechanized proofs (i.e., proofs of correctness utilizing a theorem prover) of both Grover's quantum search algorithm and the QPE algorithm. Two conceptual challenges arise in the work: First, the quantum algorithms need to be manually translated into QHL, which seems a significant effort, the result of which can be obfuscating of the original sense of the algorithm, and second, the formal verification appears to rely on extensive calculations of functions of matrices, with the calculations being off-loaded to PYTHON's NUMPY and SYMPY packages. The proof outputs also do not seem to be human readable. Quoting Hietala *et al.* [40], this is "then only a partial verification effort." Also, they do not prove any precision guarantees on the phase estimation in QPE. Using SQIR, which they describe as "a simple quantum language deeply embedded in the COQ proof assistant," Hietala *et al.* [40] provide proofs of correctness for a number of quantum algorithms, such as quantum teleportation, the Deutsch-Jozsa algorithm [10], Simon's algorithm [32], the quantum Fourier transform, quantum phase estimation, and Grover's algorithm [16].

It is possible to make direct comparisons with COQ proofs for the first two of our high-level statements about the QPE output. Both `qpe_exact` (31) and `qpe_best_guarantee` (38) were essentially proven in Ref. [40] and can be found in their SQIR repository. There are immediate differences in



```
In [4]: # the induction theorem for positive naturals
        fold_forall_natural_pos
```

$$\vdash \forall_P \left(\left(P(1) \land \left[\forall_{m \in \mathbb{N}^+ \,|\, P(m)}\ P(m+1)\right]\right) \Rightarrow \left[\forall_{n \in \mathbb{N}^+}\ P(n)\right]\right)$$

```
In [8]: _psi_t_def
```

$$\vdash$$
$$\forall_{t \in \mathbb{N}^+} \left(|\psi_t\rangle = \left(\left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot 2^{t-1}\cdot\varphi} \cdot |1\rangle\right)\right)\right) \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot 2^{t-2}\cdot\varphi} \cdot |1\rangle\right)\right)\right) \otimes \ldots \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot 2^{0}\cdot\varphi} \cdot |1\rangle\right)\right)\right)\right)\right)$$

```
In [13]: # finish off the Base Case
         base_case_jdgmt = sum_0_to_1_processed_01.sub_left_side_into(psi_1_def)
```

base_case_jdgmt: $\vdash |\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle\right)\right)\right)$

```
In [17]: summation_partition_01 = (
             inductive_step.instance_expr.rhs.operands[1]
             .partitioning(desired_domain.upper_bound))
```

summation_partition_01: $t \in \mathbb{N}^+ \vdash$
$$\left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right) = \left(\left(\sum_{k=0}^{2^t-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right) + \left(\sum_{k=2^t}^{(2\cdot 2^t)-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right)\right)$$

```
In [18]: summation_partition_02 = (summation_partition_01.inner_expr().rhs.
                                   operands[1].shift(Neg(Exp(two, t))))
```

summation_partition_02: $t \in \mathbb{N}^+ \vdash$
$$\left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right) = \left(\left(\sum_{k=0}^{2^t-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right) + \left(\sum_{k=0}^{2^t-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot(k+2^t)} \cdot |k+2^t\rangle_{t+1}\right)\right)\right)$$

```
In [24]: prepend_num_ket_with_one_ket_inst = prepend_num_ket_with_one_ket.instantiate(
                 {n: t, k: k},
                 assumptions=[*defaults.assumptions, InSet(k, Interval(zero, subtract(Exp(two, t), one)))])
```

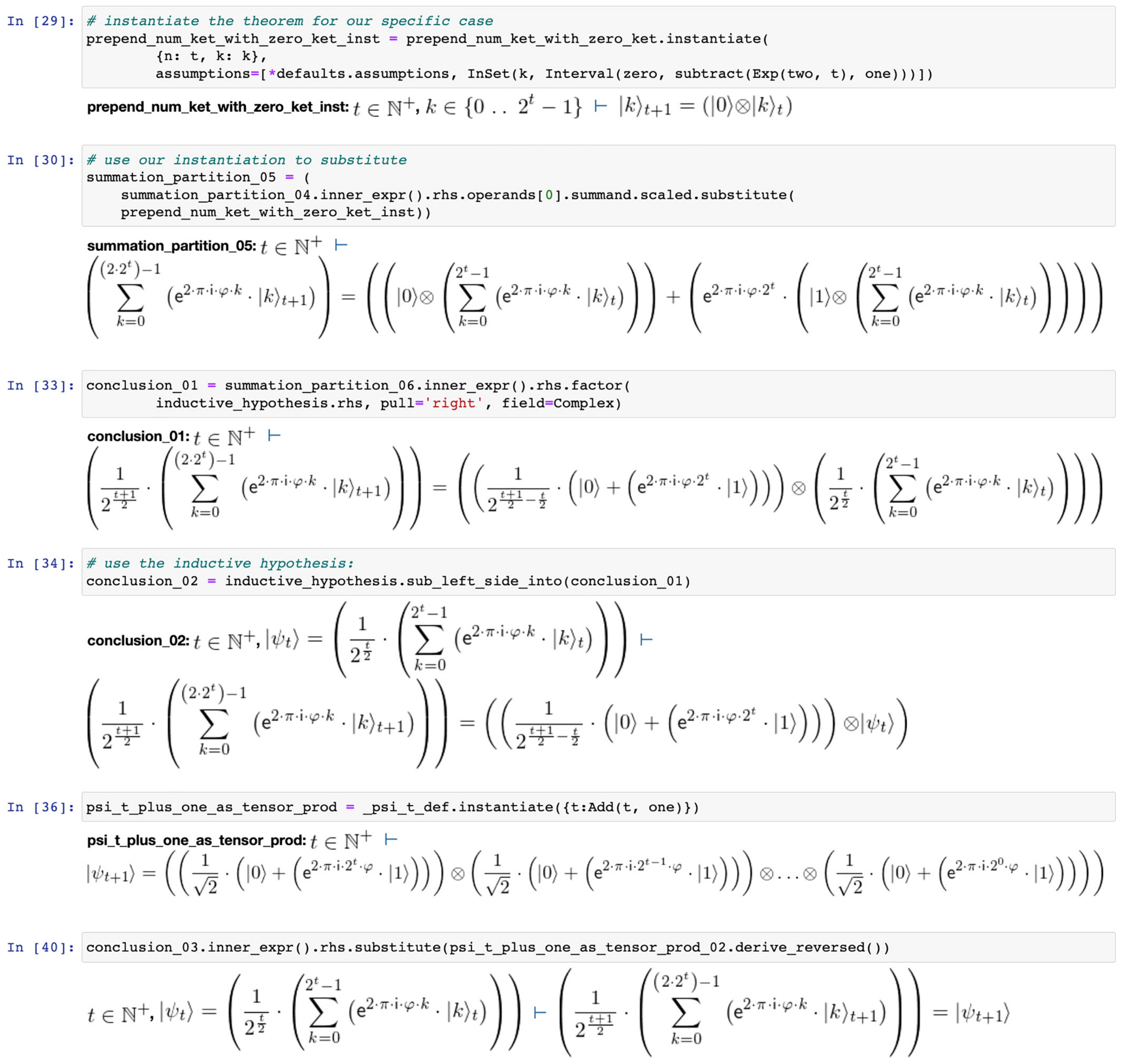
prepend_num_ket_with_one_ket_inst: $t \in \mathbb{N}^+, k \in \{0 \,..\, 2^t - 1\} \vdash |2^t + k\rangle_{t+1} = (|1\rangle \otimes |k\rangle_t)$

```
In [27]: summation_partition_04 = (
             summation_partition_02.inner_expr().rhs.operands[1].summand.substitute(
                 summand_processed_03))
```

summation_partition_04: $t \in \mathbb{N}^+ \vdash$
$$\left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right) = \left(\left(\sum_{k=0}^{2^t-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right) + \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot 2^t} \cdot \left(|1\rangle \otimes \left(\sum_{k=0}^{2^t-1} \left(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi\cdot k} \cdot |k\rangle_t\right)\right)\right)\right)\right)$$

FIG. 17. Subset of the interactive steps (13 of 23, split across Figs. 17 and 18) that construct the proof of `_psi_t_formula` (16), chosen to show the basic flow of this inductive proof and demonstrate some useful derivation commands in action such as `partitioning`, `shift`, `substitute`, and `factor`. The Appendix compares this to a structured proof format intended to communicate a proof among experts.

the way these statements are presented. Their quantum algorithm definition and statements, by necessity, are expressed as computer code, which requires knowledge of COQ to fully parse and understand. In addition, while their paper presents quantum circuits to describe algorithms, these are human translations that are not directly tied to or produced by their definitions in COQ. In contrast, our QPE specifications in terms of quantum circuits are generated directly from our definition in PROVE-IT. Our high-level QPE output statements, as expressed above, were generated directly from the proven



```
In [29]: # instantiate the theorem for our specific case
         prepend_num_ket_with_zero_ket_inst = prepend_num_ket_with_zero_ket.instantiate(
                 {n: t, k: k},
                 assumptions=[*defaults.assumptions, InSet(k, Interval(zero, subtract(Exp(two, t), one)))])
```

prepend_num_ket_with_zero_ket_inst: $t \in \mathbb{N}^+, k \in \{0 \,..\, 2^t - 1\} \vdash |k\rangle_{t+1} = (|0\rangle \otimes |k\rangle_t)$

```
In [30]: # use our instantiation to substitute
         summation_partition_05 = (
             summation_partition_04.inner_expr().rhs.operands[0].summand.scaled.substitute(
             prepend_num_ket_with_zero_ket_inst))
```

summation_partition_05: $t \in \mathbb{N}^+ \vdash$

$$\left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right) = \left(\left(|0\rangle \otimes \left(\sum_{k=0}^{2^t-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_t\right)\right)\right) + \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot 2^t} \cdot \left(|1\rangle \otimes \left(\sum_{k=0}^{2^t-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_t\right)\right)\right)\right)\right)$$

```
In [33]: conclusion_01 = summation_partition_06.inner_expr().rhs.factor(
                 inductive_hypothesis.rhs, pull='right', field=Complex)
```

conclusion_01: $t \in \mathbb{N}^+ \vdash$

$$\left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right)\right) = \left(\left(\frac{1}{2^{\frac{t+1}{2}-\frac{t}{2}}} \cdot \left(|0\rangle + \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot 2^t} \cdot |1\rangle\right)\right)\right) \otimes \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_t\right)\right)\right)\right)$$

```
In [34]: # use the inductive hypothesis:
         conclusion_02 = inductive_hypothesis.sub_left_side_into(conclusion_01)
```

conclusion_02: $t \in \mathbb{N}^+, |\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_t\right)\right)\right) \vdash$

$$\left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right)\right) = \left(\left(\frac{1}{2^{\frac{t+1}{2}-\frac{t}{2}}} \cdot \left(|0\rangle + \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot 2^t} \cdot |1\rangle\right)\right)\right) \otimes |\psi_t\rangle\right)$$

```
In [36]: psi_t_plus_one_as_tensor_prod = _psi_t_def.instantiate({t:Add(t, one)})
```

psi_t_plus_one_as_tensor_prod: $t \in \mathbb{N}^+ \vdash$

$$|\psi_{t+1}\rangle = \left(\left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2\cdot\pi\cdot i\cdot 2^t\cdot\varphi} \cdot |1\rangle\right)\right)\right) \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2\cdot\pi\cdot i\cdot 2^{t-1}\cdot\varphi} \cdot |1\rangle\right)\right)\right) \otimes \ldots \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(e^{2\cdot\pi\cdot i\cdot 2^0\cdot\varphi} \cdot |1\rangle\right)\right)\right)\right)$$

```
In [40]: conclusion_03.inner_expr().rhs.substitute(psi_t_plus_one_as_tensor_prod_02.derive_reversed())
```

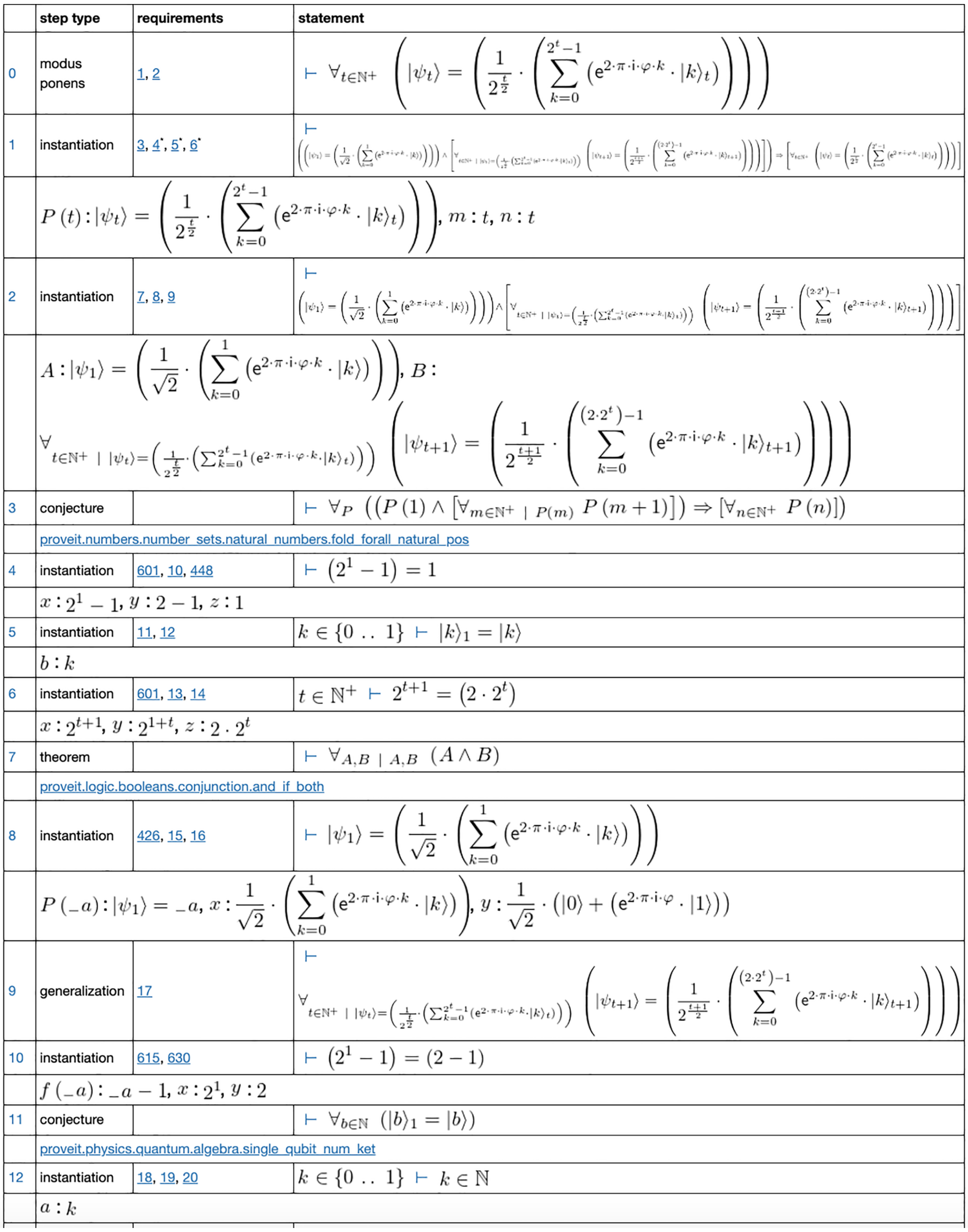
$$t \in \mathbb{N}^+, |\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_t\right)\right)\right) \vdash \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2\cdot 2^t)-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot |k\rangle_{t+1}\right)\right)\right) = |\psi_{t+1}\rangle$$

FIG. 18. Continuation from Fig. 17 showing a subset of the interactive steps that construct the proof of `_psi_t_formula` (16).

representations in PROVE-IT. There is an essentially direct correspondence between the statements that are certified by PROVE-IT and their automatically generated LATEX representations. The specification of `qpe_best_guarantee` (38) in SQIR appears in Fig. 21(a); the specification in PROVE-IT, including the automatically generated graphical circuit representation, appears in Fig. 21(c). PROVE-IT's graphical representation of the specification makes the theorem more readily comprehensible.

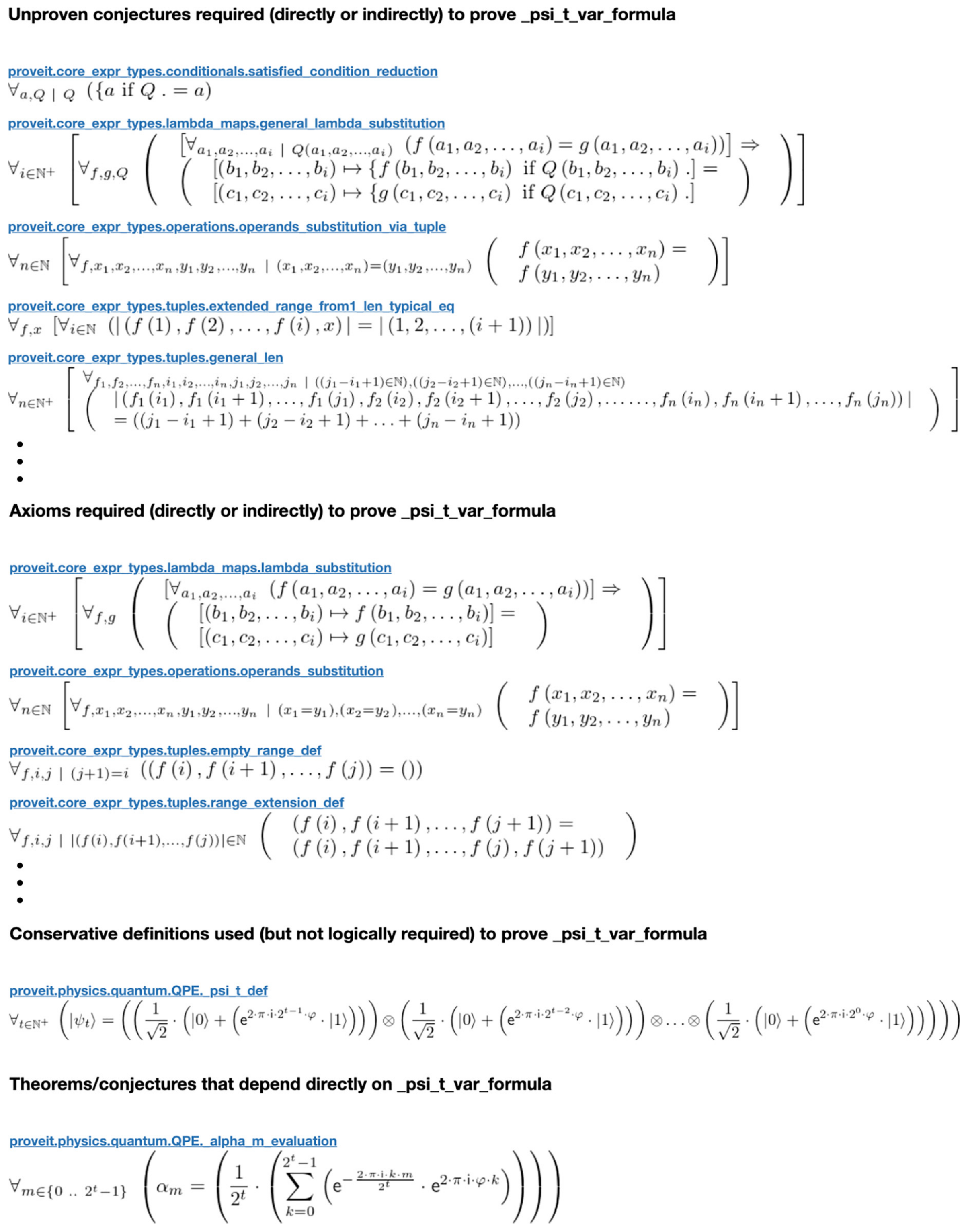
We can also compare the level of effort required to generate the proofs in COQ versus PROVE-IT. Although PROVE-IT does not have COQ's maturity and thus a significant part of the current effort in PROVE-IT is devoted to its infrastructure, one helpful comparison to make is in the number of user-invoked derivation commands in PROVE-IT versus COQ commands (e.g., tactic invocations) necessary to generate the proofs at the high level (i.e., taking more fundamental packages for granted).[9] This comparison is presented in Table IV. We can-

[9]COQ commands were enumerated for the `QPE_simplify`, `QPE_semantics_simplified`, and `QPE_full_semantics` lemmas of Hietala *et al.* [40], the most appropriate comparison to the analogous QPE-specific proofs in PROVE-IT. The number of COQ commands was estimated by simply counting all commands that ended in a period or semicolon. The PROVE-IT derivation commands tally the numbers in Tables II and III according to the dependencies of the three high-level statements respectively as shown in Fig. 8.



	step type	requirements	statement
0	modus ponens	1, 2	$\vdash \forall_{t \in \mathbb{N}^+} \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t - 1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_t\right)\right)\right)\right)$
1	instantiation	3, 4*, 5*, 6*	$\vdash \left(\left(\lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle\right)\right)\right)\right) \wedge \left[\forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2 \cdot 2^t)-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)\right]\right) \Rightarrow \left[\forall_{t \in \mathbb{N}^+} \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_t\right)\right)\right)\right)\right]$
	$P(t) : \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_t\right)\right)\right), m : t, n : t$		
2	instantiation	7, 8, 9	$\vdash \left(\lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle\right)\right)\right)\right) \wedge \left[\forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2 \cdot 2^t)-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)\right]$
	$A : \lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle\right)\right)\right), B : \forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2 \cdot 2^t)-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)$		
3	conjecture		$\vdash \forall_P \left(\left(P(1) \wedge \left[\forall_{m \in \mathbb{N}^+ \mid P(m)} P(m+1)\right]\right) \Rightarrow \left[\forall_{n \in \mathbb{N}^+} P(n)\right]\right)$
	proveit.numbers.number_sets.natural_numbers.fold_forall_natural_pos		
4	instantiation	601, 10, 448	$\vdash \left(2^1 - 1\right) = 1$
	$x : 2^1 - 1, y : 2 - 1, z : 1$		
5	instantiation	11, 12	$k \in \{0 \,..\, 1\} \vdash \lvert k\rangle_1 = \lvert k\rangle$
	$b : k$		
6	instantiation	601, 13, 14	$t \in \mathbb{N}^+ \vdash 2^{t+1} = \left(2 \cdot 2^t\right)$
	$x : 2^{t+1}, y : 2^{1+t}, z : 2 \cdot 2^t$		
7	theorem		$\vdash \forall_{A, B \mid A, B} (A \wedge B)$
	proveit.logic.booleans.conjunction.and_if_both		
8	instantiation	426, 15, 16	$\vdash \lvert\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle\right)\right)\right)$
	$P(\_a) : \lvert\psi_1\rangle = \_a, x : \frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle\right)\right), y : \frac{1}{\sqrt{2}} \cdot \left(\lvert 0\rangle + \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi} \cdot \lvert 1\rangle\right)\right)$		
9	generalization	17	$\vdash \forall_{t \in \mathbb{N}^+ \mid \lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_t\right)\right)\right)} \left(\lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}} \cdot \left(\sum_{k=0}^{(2 \cdot 2^t)-1} \left(\mathsf{e}^{2 \cdot \pi \cdot \mathsf{i} \cdot \varphi \cdot k} \cdot \lvert k\rangle_{t+1}\right)\right)\right)\right)$
10	instantiation	615, 630	$\vdash \left(2^1 - 1\right) = (2 - 1)$
	$f(\_a) : \_a - 1, x : 2^1, y : 2$		
11	conjecture		$\vdash \forall_{b \in \mathbb{N}} \left(\lvert b\rangle_1 = \lvert b\rangle\right)$
	proveit.physics.quantum.algebra.single_qubit_num_ket		
12	instantiation	18, 19, 20	$k \in \{0 \,..\, 1\} \vdash k \in \mathbb{N}$
	$a : k$		

FIG. 19. First few lines of PROVE-IT's formal proof output for `_psi_t_formula` (16), illustrating both the general structure of a formal proof in PROVE-IT and the rigorous and explicitly human-checkable details involved in each proof step (the full proof has 684 steps at this time). The steps appear in reverse order with the proven statement at the top as the root of the proof DAG. Each numbered step (along the left-hand side) includes a step type and an explicit listing of all previous (higher-numbered) steps on which the current step depends (note the underlined numbers in the "requirements" column, always higher numbered than the current step to indicate they occur further down the proof tree with no circular dependency). The underlined numbers are hyperlinks that will take the user to the corresponding step in the formal proof. Instantiation steps (such as step 4) explicitly list the substitutions used in the derivation. Steps that introduce an assumption, axiom, theorem, or conjecture (such as step 7) show the detailed formula being introduced, as well as its name in the PROVE-IT database hyperlinked to its proof (or the placeholder for the proof if one does not yet exist).



Unproven conjectures required (directly or indirectly) to prove _psi_t_var_formula

proveit.core_expr_types.conditionals.satisfied_condition_reduction

$$\forall_{a,Q \,|\, Q}\ (\{a \text{ if } Q\ . = a)$$

proveit.core_expr_types.lambda_maps.general_lambda_substitution

$$\forall_{i \in \mathbb{N}^+}\left[\forall_{f,g,Q}\left(\begin{array}{l}\left[\forall_{a_1,a_2,\ldots,a_i \,|\, Q(a_1,a_2,\ldots,a_i)}\ (f(a_1,a_2,\ldots,a_i) = g(a_1,a_2,\ldots,a_i))\right] \Rightarrow \\ \left(\begin{array}{l}[(b_1,b_2,\ldots,b_i) \mapsto \{f(b_1,b_2,\ldots,b_i) \text{ if } Q(b_1,b_2,\ldots,b_i)\ .] = \\ {[(c_1,c_2,\ldots,c_i) \mapsto \{g(c_1,c_2,\ldots,c_i) \text{ if } Q(c_1,c_2,\ldots,c_i)\ .]}\end{array}\right)\end{array}\right)\right]$$

proveit.core_expr_types.operations.operands_substitution_via_tuple

$$\forall_{n \in \mathbb{N}}\left[\forall_{f,x_1,x_2,\ldots,x_n,y_1,y_2,\ldots,y_n \,|\, (x_1,x_2,\ldots,x_n)=(y_1,y_2,\ldots,y_n)}\left(\begin{array}{l}f(x_1,x_2,\ldots,x_n) = \\ f(y_1,y_2,\ldots,y_n)\end{array}\right)\right]$$

proveit.core_expr_types.tuples.extended_range_from1_len_typical_eq

$$\forall_{f,x}\ [\forall_{i \in \mathbb{N}}\ (|(f(1),f(2),\ldots,f(i),x)| = |(1,2,\ldots,(i+1))|)]$$

proveit.core_expr_types.tuples.general_len

$$\forall_{n \in \mathbb{N}^+}\left[\begin{array}{l}\forall_{f_1,f_2,\ldots,f_n,i_1,i_2,\ldots,i_n,j_1,j_2,\ldots,j_n \,|\, ((j_1-i_1+1)\in\mathbb{N}),((j_2-i_2+1)\in\mathbb{N}),\ldots,((j_n-i_n+1)\in\mathbb{N})} \\ \left(\begin{array}{l}|(f_1(i_1),f_1(i_1+1),\ldots,f_1(j_1),f_2(i_2),f_2(i_2+1),\ldots,f_2(j_2),\ldots\ldots,f_n(i_n),f_n(i_n+1),\ldots,f_n(j_n))| \\ = ((j_1-i_1+1)+(j_2-i_2+1)+\ldots+(j_n-i_n+1))\end{array}\right)\end{array}\right]$$

•
•
•

Axioms required (directly or indirectly) to prove _psi_t_var_formula

proveit.core_expr_types.lambda_maps.lambda_substitution

$$\forall_{i \in \mathbb{N}^+}\left[\forall_{f,g}\left(\begin{array}{l}[\forall_{a_1,a_2,\ldots,a_i}\ (f(a_1,a_2,\ldots,a_i) = g(a_1,a_2,\ldots,a_i))] \Rightarrow \\ \left(\begin{array}{l}[(b_1,b_2,\ldots,b_i) \mapsto f(b_1,b_2,\ldots,b_i)] = \\ {[(c_1,c_2,\ldots,c_i) \mapsto g(c_1,c_2,\ldots,c_i)]}\end{array}\right)\end{array}\right)\right]$$

proveit.core_expr_types.operations.operands_substitution

$$\forall_{n \in \mathbb{N}}\left[\forall_{f,x_1,x_2,\ldots,x_n,y_1,y_2,\ldots,y_n \,|\, (x_1=y_1),(x_2=y_2),\ldots,(x_n=y_n)}\left(\begin{array}{l}f(x_1,x_2,\ldots,x_n) = \\ f(y_1,y_2,\ldots,y_n)\end{array}\right)\right]$$

proveit.core_expr_types.tuples.empty_range_def

$$\forall_{f,i,j \,|\, (j+1)=i}\ ((f(i),f(i+1),\ldots,f(j)) = ())$$

proveit.core_expr_types.tuples.range_extension_def

$$\forall_{f,i,j \,|\, |(f(i),f(i+1),\ldots,f(j))| \in \mathbb{N}}\left(\begin{array}{l}(f(i),f(i+1),\ldots,f(j+1)) = \\ (f(i),f(i+1),\ldots,f(j),f(j+1))\end{array}\right)$$

•
•
•

Conservative definitions used (but not logically required) to prove _psi_t_var_formula

proveit.physics.quantum.QPE._psi_t_def

$$\forall_{t \in \mathbb{N}^+}\left(|\psi_t\rangle = \left(\left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot 2^{t-1}\cdot\varphi} \cdot |1\rangle\right)\right)\right) \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot 2^{t-2}\cdot\varphi} \cdot |1\rangle\right)\right)\right) \otimes \ldots \otimes \left(\frac{1}{\sqrt{2}} \cdot \left(|0\rangle + \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot 2^{0}\cdot\varphi} \cdot |1\rangle\right)\right)\right)\right)\right)$$

Theorems/conjectures that depend directly on _psi_t_var_formula

proveit.physics.quantum.QPE._alpha_m_evaluation

$$\forall_{m \in \{0\ ..\ 2^t-1\}}\left(\alpha_m = \left(\frac{1}{2^t} \cdot \left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{-\frac{2\cdot\pi\cdot\mathsf{i}\cdot k\cdot m}{2^t}} \cdot \mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\right)\right)\right)\right)$$

FIG. 20. Excerpt from the dependencies page for `_psi_t_formula` (16), showing some of the unproven theorems, axioms, and conservative definitions on which the theorem depends, and `_alpha_m_evaluation` (27), which depends directly on the `_psi_t_formula` theorem itself. Such dependencies are all explicitly tracked in PROVE-IT and readily accessible for review by the user.

not make a comparison for the most challenging proof since this particular form was not proven in Ref. [40]. Such numbers demonstrate that the interactive PROVE-IT process is quite competitive with the effort required in the COQ proof assistant, in addition to PROVE-IT also producing a human-readable formal proof for one's effort: 87 PROVE-IT derivation commands versus 137 COQ commands for the `qpe_exact` theorem, and 213 versus 313 for the `qpe_best_guarantee` theorem.

Amy used the Feynman path integral model of quantum circuits for their optimization, synthesis, and functional

(a)

```
Lemma QPE_semantics_full : forall k n (c : base_ucom n) z (ψ : Vector (2 ^ n)) (δ : R),
  (n > 0)%nat -> (k > 1)%nat -> uc_well_typed c -> Pure_State_Vector ψ ->
  (-1 / 2 ^ (k + 1) <= δ < 1 / 2 ^ (k + 1))%R ->
  let θ := ((INR (funbool_to_nat k z) / 2 ^ k) + δ)%R in
  (uc_eval c) × ψ = Cexp (2 * PI * θ) .* ψ ->
  probability_of_outcome
      ((f_to_vec k z) ⊗ ψ)
      (@Mmult _ _ (1 * 1) (uc_eval (QPE k n c)) (k ⨂ ∣0⟩ ⊗ ψ))
  >= 4 / (PI ^ 2).
```

(b)

```
let  ghost function qpe_meas (circ:circuit) (k n:int)(y:matrix complex) ( theta:complex): complex
                requires{n= k+ width circ}
                requires{real_ theta}
                requires{1 <= k<n}
                requires{is_a_ket_l y (n-k)}
                requires{eigen circ y (real_to_ang theta)}
                requires{c_zero <. theta <. c_one}
                ensures{delta theta k = c_zero -> result =   c_one}
                ensures{c_zero <. delta theta k ->  i_to_c 4 /. cpower pi 2 <=. result}
  = if delta theta k == c_zero then
      ind_sum_constant_from_z  (fun y -> (cpower (ang_exp (real_to_ang (delta theta k))) y)) c_one  (power_ 2 k)
    else geom_sum_bound (delta theta k) (power_ 2 k);
    assert{(real_to_ang (delta theta k)) = real_to_ang (theta  -.  (i_to_c   (best_appr theta k)  *. pow_inv_2 k))};
    pe_measure circ k (mod (best_appr theta k) (power_ 2 k)) n  y theta
end
```

(c)

```
qpe_best_guarantee = (
    Forall((s, t), Forall(U, Forall(var_ket_u, Forall(phase,
        greater(Prob(_phase_est_circuit_b_r.literals_as_variables(_s, _t, _U, _ket_u, _phase)),
                frac(four, Exp(pi, two))),
        domain=Real,
        conditions=[_phase_in_interval.literals_as_variables(_phase),
                    Equals(MatrixMult(U, var_ket_u)),
                    ScalarMult(exp2pi_i(phase), var_ket_u))]).with_wrapping(),
    domain=s_ket_domain, condition=normalized_var_ket_u),
    domain=Unitary(two_pow_s)),
    domain=NaturalPos))
```

qpe_best_guarantee (conjecture with conjecture-based proof):

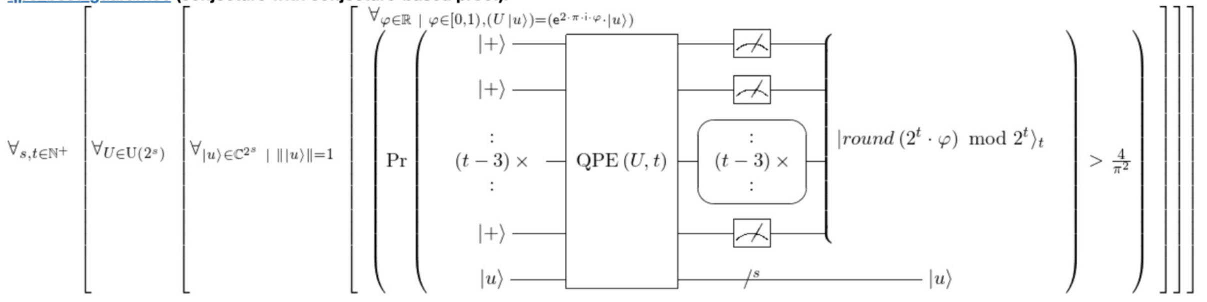

$\forall_{s,t\in\mathbb{N}^+}$
$\forall_{U\in\mathrm{U}(2^s)}$
$\forall_{|u\rangle\in\mathbb{C}^{2^s}\ |\ \||u\rangle\|=1}$
$\forall_{\varphi\in\mathbb{R}\ |\ \varphi\in[0,1),(U\,|u\rangle)=(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi}\cdot|u\rangle)}$
Pr
$|+\rangle$
$|+\rangle$
$(t-3)\times$
$|+\rangle$
$|u\rangle$
$\mathrm{QPE}(U,t)$
$(t-3)\times$
s
$|round\,(2^t\cdot\varphi)\ \mathrm{mod}\ 2^t\rangle_t$
$|u\rangle$
$> \frac{4}{\pi^2}$

FIG. 21. Specifications of qpe_best_guarantee (38) in (a) SQIR [40] vs (b) QBricks [15] vs (c) PROVE-IT. The specification in PROVE-IT automatically generates the accompanying circuit representation, contributing to clarity of presentation and human readability.

TABLE IV. Number of PROVE-IT derivation command steps vs COQ commands [40] for the main QPE theorems. The "reused" and "additional" steps sum to give the left-column total, providing a measure of how many derivation steps in later theorems are reused from earlier theorems.

	PROVE-IT derivation commands			COQ commands		
Theorem	Total	Reused	Additional	Total	Reused	Additional
qpe_exact	87			137		
qpe_best_guarantee	213	72	141	313	73	240
qpe_precision_guarantee	437	145	292			

verification [52]. It particularly focused on the functional verification of quantum circuits built using Clifford+R_k gates, where $R_k = R_z(\frac{2\pi}{2^k})$ and R_z performs rotation around the z axis. A semantic model of such Clifford+R_z circuits as path sums is developed along with a calculus of rewrite rules; these circuits are verified against a human-readable `input/output` specification. It is shown there that path sums associated with Clifford+R_z circuits can be represented by a collection of path variables, a pseudo-Boolean phase function, and an affine basis transformation, whereas a pseudo-Boolean phase function has a unique representation as a multilinear polynomial over variables. A calculus of rewrite rules over polynomial representations is used to compute a normal-form polynomial associated with a quantum circuit that can be computed in polynomial time; these normal forms are however not unique. Quantum algorithms or circuits for reversible functions, particularly implementations of Toffoli gates and adders using Clifford+T gates, were verified; more nontrivial algorithms including quantum Fourier transform and quantum hidden-shift algorithms were also verified. Verification thus amounts to checking for circuit equivalence performed by computing normal forms of circuits. The method is implemented in a quantum compiler backend infrastructure FEYNMAN for quantum circuits, similar to LLVM for programming languages [53,54]. Since Amy's work does not consider measurement gates, precision guarantees on phase estimation are omitted.

Using their QBRICKS quantum circuit-building program environment [25] within the WHY3 deductive verification environment [55], Chareton *et al.* [24] encode and verify implementations of Grover's quantum search algorithm, the quantum Fourier transform, quantum phase estimation, and Shor's algorithm. They use Amy's Feynman path sum model of quantum circuits as building blocks or program constructs, which allows other quantum algorithms to be written as finite branching straight-line programs in a Floyd-Hoare axiomatic framework for which WHY3 is well suited for generating the associated verification conditions for input-output behavioral verification. They show that their approach produces verifications with high automaticity and high efficiency (in terms of the size of the algorithm specification and the number of commands required in the verification effort) compared to similar proofs in SQIR and QHL. Like other approaches, they have built a collection of theories which enable reasoning about sets, algebra, arithmetic, iterators, quantum data, Kronecker products, unity circles, etc. They also generalized Amy's work to implement parametrized path sums for modeling quantum circuits with arbitrary numbers of qubits.

The QBRICKS approach provides formal verification with high automaticity, whereas PROVE-IT facilitates the generation of a human-readable formal proof based on user-interactive informal proof steps. As with the use of SQIRby Hietala *et al.* [40], the algorithm specification in QBRICKS is more difficult to grasp compared to the specification in PROVE-IT. The QBRICKS specification for the exact and general QPE results appears in Fig. 21(b). Moreover, the interactive steps in PROVE-IT, understandably missing from a system such as QBRICKS dedicated to high automation and rather obscure in COQ code, can provide insights, as evidenced by this process uncovering a number of minor errors in the classic Nielsen-Chuang presentation [31], which we describe in Sec. VII. More generally, we would argue that PROVE-IT offers advantages in flexibility and relative ease of use over the QBRICKS and SQIR or COQ approaches and compares favorably in terms of time and effort even as it produces a human-readable formal proof largely lacking in the other systems.

VII. DISCUSSION AND CONCLUSIONS

Current quantum computers are not powerful enough, in size or fidelity, to test ideas beyond the smallest demonstrations. As they become more powerful, debugging quantum programs through direct probing will be an increasing challenge due to the complex nature of quantum information: An arbitrary state of n qubits must be described using 2^n complex numbers and probing of circuit states results in probabilistic outcomes. Formal program verification offers an alternative approach to helping ensure correct quantum programs.

In this paper we demonstrated an alternative approach to formal theorem proving applied to the verification of an important quantum algorithm with an exponential speed advantage in comparison to the best known classical algorithm for accomplishing the same feat, quantum phase estimation. There is significant value in being able to formally verify quantum algorithms and their implementations in order to guide investments and prevent hard-to-find bugs in quantum programs. The JUPYTER notebook-based interface and underlying PYTHON code make the PROVE-IT system accessible to a wide audience, moderately easy to learn and use, and relatively easy to make contributions to its expandable knowledge base of axioms, conjectures, theorems, proofs, and proof tactics.

PROVE-IT's approach is unique by presenting mathematical or technical statements, allowing a user to derive new statements in a fashion that mimics informal theorem proving, and finally presenting a full airtight proof that is human readable and machine checkable. In this paper we demonstrated these features as applied to proving three successively sophisticated high-level statements about the output distribution of the QPE algorithm. We showed our decomposition of these proofs into several supporting theorems (lemmas) that parallel a textbook proof [31]. We then explained the guarantees that PROVE-IT provides, by tracking dependencies, ensuring there is no circular logic, and certifying each proof with understandable, complete, and machine-checkable derivations. After describing how our proofs were generated with the use of automation features to fill in obvious steps, making formal theorem proving nearly as straightforward as constructing an informal but rigorous proof, we focused on our proof for one of the QPE-specific lemmas as a demonstration. Finally, we made comparisons with QPE proofs in other theorem-proving systems and noted that PROVE-IT is uniquely versatile in using broadly understood and convenient mathematical or diagrammatic notation for human-readable theorems and proofs. We also noted that we compare favorably against a COQ implementation for proving QPE in the number of derivation or tactic commands that were used in our respective proofs.

It is interesting to note that the interactive proving process within PROVE-IT's theorem notebooks helped uncover several typographical errors in the original text [31] and eventually led to a better bound on the precision guarantee. For example, in their Sec. 5.2.1 on performance and requirements, Nielsen and Chuang cite the constraint $0 \leqslant \delta \leqslant 2^{-t}$ whereas $0 \leqslant \delta < 2^{-t}$ (with the strict inequality on the right) is actually needed, and they use $-\pi \leqslant 2\pi(\delta - \ell/2^t) \leqslant \pi$ where $-\pi \leqslant 2\pi(\delta - \ell/2^t) < \pi$ (again with the strict inequality on the right) is actually needed. There is also an absolute value missing on the right-hand side of Nielsen and Chuang's Eq. (5.29), and the equal sign in their Eq. (5.34) should be a less-than sign. We were also able to process the summations slightly differently in their Eq. (5.31) to eventually arrive at a better bound on the probability of a measurement failure, obtaining the tighter $p(|m-b| > e) < \frac{1}{2e} + \frac{1}{4e^2}$ instead of $\frac{1}{2(1-e)}$ and allowing $e \geqslant 1$ instead of just $e \geqslant 2$.

There is much work remaining to be done in developing the PROVE-IT system to meet its potential as a powerful and valuable theorem-proving system. The present work demonstrates what is possible with this approach. However, in this instance, it did require a concerted effort by a small team to implement specific capabilities. Ultimately, we envision a system able to incorporate features developed by a diverse range of contributors to support the needs of an even broader audience of users for a variety of applications. With more contributions, the system would become more powerful and garner further contributions in a positive feedback loop. Before reaching that critical transition, there is more work to be done by a small, dedicated team. Tutorials and package "demonstration" pages need to be completed so people can learn how to use the system and make contributions, a number of known software issues need to be addressed, and processes need to be improved for handling higher throughputs.

All of the code, definitions, axioms, lemmas, and theorems discussed in this article in relation to the QPE verification efforts are openly available at a version of the PROVE-IT website archived specifically for this paper [37]. The most current demonstration version of PROVE-IT can be accessed at its public website [33] and all current development code is available at the PROVE-IT GitHub site [34].

ACKNOWLEDGMENTS

We gratefully acknowledge Kenny Rudinger and Mohan Sarovar, who contributed substantially to an earlier proof-of-concept demonstration of using PROVE-IT to verify the QPE algorithm [56] and for their continued support and encouragement. We acknowledge Joaquín E. Madrid Larrañaga, who contributed to a number of PROVE-IT features (including the generation of quantum circuit LATEX, basic arithmetic, etc.), testing, and documentation. We also thank Jon Aytac, Geoffrey Hulette, Denis Bueno, and Geoffrey Reedy for valuable discussions and contributions to the development of PROVE-IT, Kesha Hietala for generous help in accessing and understanding SQIR and related COQ material, and Christophe Chareton for generous help in accessing and understanding QBRICKS. This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the Quantum Computing Applications Team and Quantum Systems Accelerator programs, and the Laboratory Directed Research and Development (LDRD) program at Sandia National Laboratories under LDRD Project No. 15-2618. Sandia National Laboratories is a multiprogram laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract No. DE-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the U.S. Government.

APPENDIX: STRUCTURED PROOF EXAMPLE

Writing structured proofs that clearly and concisely communicate concepts and proof strategies to a community of experts is an important goal in its own right which does not require a sophisticated formal theorem-proving software system [44,45]. Ideally, a proof is verified in a sophisticated software system like PROVE-IT and is also presentable for clearly communicating concepts. PROVE-IT's origins drew inspiration from work by one of our authors (R.C.) in developing a system to format structured proofs by hand. His proof via dependencies format first lists a collection of accepted relevant mathematical facts in a familiar notation as "Assumptions." Then each proof step begins with a conjunction of explicit assumptions and results from previous proof steps and ends with the keyword "Thus" and a conclusion based on that specific collection of assumptions and previous step results. The following is a demonstration of using this format to prove `_psi_t_formula` (16) (and the focus of Sec. V).

Theorem. If $\forall_{t\in\mathbb{N}^+}$, $\psi_t = \frac{1}{2^{t/2}}[(|0\rangle + e^{2\pi i 2^{t-1}\varphi}|1\rangle) \otimes (|0\rangle + e^{2\pi i 2^{t-2}\varphi}|1\rangle) \otimes (|0\rangle + e^{2\pi i 2^{t-2}\varphi}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i 2^{0}\varphi}|1\rangle)]$, then $\forall_{t\in\mathbb{N}^+}$, $\psi_t = \frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} e^{2\pi i \varphi k}|k\rangle_t$.

Assumptions. We make the following assumptions.

(1) $\varphi \in [0, 1)$.

(2) $\forall_{k\in]t[}$, $p_k = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 2^k \varphi}|1\rangle)$, where $]t[= \{0, 1, 2, \ldots, t-1\}$.

(3) $\psi_1 = p_0$ and $\forall_{k\in[t-1]}$, $\psi_{k+1} = p_k \otimes \psi_k$, e.g., $\psi_t = p_{t-1} \otimes p_{t-2} \otimes \cdots \otimes p_0$.

(4) $\forall_{k\in]t[}$, $p'_k = (|0\rangle + e^{2\pi i 2^k \varphi}|1\rangle)$.

(5) $\psi'_1 = p'_0$ and $\forall_{k\in[t-1]}$, $\psi'_{k+1} = p'_k \otimes \psi'_k$.

(6) $\forall_{t\in\mathbb{N}^+}, \forall_{k\in]2^t[}$, $|0\rangle \otimes |k\rangle_t = |k\rangle_{t+1}$.

(7) $\forall_{t\in\mathbb{N}^+}, \forall_{k\in]2^t[}$, $|1\rangle \otimes |k\rangle_t = |2^t + k\rangle_{t+1}$.

(8) $\forall_a, \forall_b, \forall_J$, $\sum_{j\in J} a \otimes b_j = a \otimes \sum_{j\in J} b_j$.

(9) $\sum_{k=0}^{k=t} a|k\rangle_t = a\sum_{k=0}^{k=t} |k\rangle_t$.

(10) $\psi_{t-1} = \frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} e^{2\pi i \varphi k}|k\rangle_t$ if and only if $\psi'_{t-1} = \sum_{k=0}^{2^t-1} e^{2\pi i \varphi k}|k\rangle_t$.

(11) The principle of mathematical induction: If $P(1)$ and $\forall_{t\in\mathbb{N}^+}$, $P(t) \Rightarrow P(t+1)$, then $\forall_{t\in\mathbb{N}^+}$, $P(t)$.

(12) $\forall_{t\in\mathbb{N}^+}$, $\psi_t = \frac{1}{2^{t/2}}[(|0\rangle + e^{2\pi i 2^{t-1}\varphi}|1\rangle) \otimes (|0\rangle + e^{2\pi i 2^{t-2}\varphi}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i 2^{0}\varphi}|1\rangle)]$.

(13) $\forall_{t\in\mathbb{N}^+}$, $f(t) = \frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} e^{2\pi i \varphi k}|k\rangle_t$.

(14) $\forall_{t\in\mathbb{N}^+}$, $P(t) \Leftrightarrow \psi_t = f(t)$.

(15) $\psi_1 = \frac{1}{2^{1/2}}(|0\rangle + e^{2\pi i \varphi}|1\rangle)$.

(16) $f(1) = \frac{1}{2^{1/2}}(|0\rangle_1 + e^{2\pi i \varphi}|1\rangle_1)$.

(17) $\forall_{a,b,c\in\mathbb{Z}}, \forall_\alpha$, $\sum_{k=a}^{c} \alpha_k = \sum_{k=a}^{b} \alpha_k + \sum_{k=b+1}^{c} \alpha_k$.

TABLE V. Condensed, annotated version of PROVE-IT's formal proof of `_psi_t_formula` (16), capturing the most substantive derivation steps. Each step specifies previous steps on which it depends [listed in the "Reason(s)" column], except where dependent steps have been omitted (in which case the reason is articulated in words). The "Notebook cell" numbers refer to the corresponding user-interactive inputs shown in Figs. 17 and 18 leading to the formal proof.

Step	Judgment	Reason(s)	Notebook cell
0	$\vdash \forall_{t\in\mathbb{N}^+}\{\lvert\psi_t\rangle = [\frac{1}{2^{t/2}}(\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_t))]\}$	3, 8, 17	
3	$\vdash \forall_P(\{P(1)\land[\forall_{m\in\mathbb{N}^+\lvert P(m)}P(m+1)]\}\Rightarrow[\forall_{n\in\mathbb{N}^+}P(n)])$	Induction theorem	4
8	$\vdash \lvert\psi_1\rangle = \frac{1}{\sqrt{2}}\sum_{k=0}^{1}(e^{2\pi i\varphi k}\lvert k\rangle_1)$	101	13
17	$t\in\mathbb{N}^+, \lvert\psi_t\rangle = [\frac{1}{2^{t/2}}(\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_t))]$ $\vdash \lvert\psi_{t+1}\rangle = [\frac{1}{2^{(t+1)/2}}(\sum_{k=0}^{2\times 2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1}))]$	47, 84, 101	40
47	$t\in\mathbb{N}^+, \lvert\psi_t\rangle = [\frac{1}{2^{t/2}}(\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_t))]$ $\vdash \frac{1}{2^{(t+1)/2}}(\sum_{k=0}^{2\times 2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1})) = (\frac{1}{\sqrt{2}}[\lvert 0\rangle + (e^{2\pi i\varphi 2^t}\lvert 1\rangle)])\otimes\lvert\psi_t\rangle$	60	34
60	$t\in\mathbb{N}^+ \vdash \frac{1}{2^{(t+1)/2}}(\sum_{k=0}^{2\times 2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1}))$ $= (\frac{1}{\sqrt{2}}[\lvert 0\rangle + (e^{2\pi i\varphi 2^t}\lvert 1\rangle)])\otimes[\frac{1}{2^{t/2}}(\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_t))]$	92	33
84	$t\in\mathbb{N}^+ \vdash \lvert\psi_{t+1}\rangle = (\frac{1}{\sqrt{2}}[\lvert 0\rangle + (e^{2\pi i 2^t\varphi}\lvert 1\rangle)])$ $\otimes(\frac{1}{\sqrt{2}}[\lvert 0\rangle + (e^{2\pi i 2^{t-1}\varphi}\lvert 1\rangle)])\otimes\cdots$ $\otimes(\frac{1}{\sqrt{2}}[\lvert 0\rangle + (e^{2\pi i 2^0\varphi}\lvert 1\rangle)])$	101	36
92	$t\in\mathbb{N}^+ \vdash \sum_{k=0}^{2\times 2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1})$ $= \lvert 0\rangle\otimes(\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_t))$ $+(e^{2\pi i\varphi 2^t}\lvert 1\rangle)\otimes(\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_t))$	142, 178, 181 218, 350	30
101	$\vdash \forall_{t\in\mathbb{N}^+}[\lvert\psi_t\rangle = (\frac{1}{\sqrt{2}}[\lvert 0\rangle + (e^{2\pi i 2^{t-1}\varphi}\lvert 1\rangle)])$ $\otimes(\frac{1}{\sqrt{2}}[\lvert 0\rangle + (e^{2\pi i 2^{t-2}\varphi}\lvert 1\rangle)])\otimes\cdots$ $\otimes(\frac{1}{\sqrt{2}}[\lvert 0\rangle + (e^{2\pi i 2^0\varphi}\lvert 1\rangle)])]$	Definition	8
142	$t\in\mathbb{N}^+ \vdash \sum_{k=0}^{2\times 2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1})$ $= \sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1}) + \sum_{k=0}^{2^t-1}(e^{2\pi i\varphi(k+2^t)}\lvert k+2^t\rangle_{t+1})$	173, 174	18
173	$t\in\mathbb{N}^+ \vdash \sum_{k=0}^{2\times 2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1})$ $= \sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1}) + \sum_{k=2^t}^{2\times 2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1})$	Summation split	17
174	$t\in\mathbb{N}^+ \vdash \sum_{k=2^t}^{2\times 2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_{t+1}) = \sum_{k=0}^{2^t-1}(e^{2\pi i\varphi(k+2^t)}\lvert k+2^t\rangle_{t+1})$	Summation shift	18
178	$t\in\mathbb{N}^+ \vdash \sum_{k=0}^{2^t-1}[(e^{2\pi i\varphi k}e^{2\pi i\varphi 2^t})(\lvert 1\rangle\otimes\lvert k\rangle_t)]$ $= e^{2\pi i\varphi 2^t}[\lvert 1\rangle\otimes(\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_t))]$	Linearity of tensor product	27
181	$t\in\mathbb{N}^+ \vdash \lvert 0\rangle\otimes(\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}\lvert k\rangle_t)) = \sum_{k=0}^{2^t-1}[e^{2\pi i\varphi k}(\lvert 0\rangle\otimes\lvert k\rangle_t)]$	Linearity of tensor product	30
218	$t\in\mathbb{N}^+, k\in\{0\cdots 2^t-1\} \vdash \lvert k\rangle_{t+1} = (\lvert 0\rangle\otimes\lvert k\rangle_t)$	Ket definition	29
350	$t\in\mathbb{N}^+, k\in\{0\cdots 2^t-1\} \vdash \lvert 2^t+k\rangle_{t+1} = (\lvert 1\rangle\otimes\lvert k\rangle_t)$	Ket definition	24

(18) $\forall_{k\in\{2^{t'},2^{t'}+1,\ldots,2^{t^*}-1\}}, \quad e^{2\pi i\varphi k}\lvert k\rangle_{t^*} = e^{2\pi i\varphi(k-2^{t'}+2^{t'})}\lvert k-2^{t'}+2^{t'}\rangle_{t^*}$.

Structured proof (by induction on t). For the base case we have the following.

(1) $1\in\mathbb{N}^+$, $P(1)\Leftrightarrow\psi_1 = f(1)$, $\psi_1 = \frac{1}{2^{1/2}}[(\lvert 0\rangle + e^{2\pi i\varphi}\lvert 1\rangle)]$, $f(1) = \frac{1}{2^{1/2}}[(\lvert 0\rangle_1 + e^{2\pi i\varphi}\lvert 1\rangle_1)]$, $\lvert 0\rangle = \lvert 0\rangle_1$, and $\lvert 1\rangle = \lvert 1\rangle_1$. Thus $P(1)$.

For the inductive step we have the following.

(2) Let $t'\in\mathbb{N}^+$ such that $P(t')$ and $\forall_{t\in\mathbb{N}^+}$, $P(t)\Leftrightarrow\psi_t = f(t)$. Thus, $\psi_{t'} = f(t')$.

(3) $t^* = t'+1$, $\forall_{a,b,c\in\mathbb{Z}}$ and $\forall_\alpha$, $\sum_{k=a}^{c}\alpha_k = \sum_{k=a}^{b}\alpha_k + \sum_{k=b+1}^{c}\alpha_k$. Thus $\sum_{k=0}^{2^{t^*}-1}e^{2\pi i\varphi k}\lvert k\rangle_{t^*} = \sum_{k=0}^{2^{t'}-1}e^{2\pi i\varphi k}\lvert k\rangle_{t^*} + \sum_{k=2^{t'}}^{2^{t^*}-1}e^{2\pi i\varphi k}\lvert k\rangle_{t^*}$.

(4) $\forall_{k\in\{2^{t'},2^{t'}+1,\ldots,2^{t^*}-1\}}$, $e^{2\pi i\varphi k}\lvert k\rangle_{t^*} = e^{2\pi i\varphi(k-2^{t'}+2^{t'})}\lvert k-2^{t'}+2^{t'}\rangle_{t^*}$. Thus $\sum_{k=2^{t'}}^{2^{t^*}-1}e^{2\pi i\varphi k}\lvert k\rangle_{t^*} = \sum_{k=0}^{2^{t'}-1}e^{2\pi i\varphi(k+2^{t'})}\lvert k+2^{t'}\rangle_{t^*}$.

(5) $\sum_{k=0}^{2^{t^*}-1}e^{2\pi i\varphi k}\lvert k\rangle_{t^*} = \sum_{k=0}^{2^{t'}-1}e^{2\pi i\varphi k}\lvert k\rangle_{t^*} + \sum_{k=2^{t'}}^{2^{t^*}-1}e^{2\pi i\varphi k}\lvert k\rangle_{t^*}$ and $\sum_{k=2^{t'}}^{2^{t^*}-1}e^{2\pi i\varphi k}\lvert k\rangle_{t^*} = \sum_{k=0}^{2^{t'}-1}e^{2\pi i\varphi(k+2^{t'})}$

$|k + 2^{t'}\rangle_{t^*}$. Thus $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} = \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} + \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(k+2^{t'})}|k + 2^{t'}\rangle_{t^*}$.

(6) $\forall_a$, $\forall_b$, $\forall_J$, $\sum_{j\in J} a \otimes b_j = a \otimes \sum_{j\in J} b_j$, $\forall_{t\in\mathbb{N}^+}$ and $\forall_{k\in]2^t[}$, $|1\rangle \otimes |k\rangle_t = |2^t + k\rangle_{t+1}$. Thus $\sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(2^{t'}+k)}|2^{t'} + k\rangle_{t^*} = e^{2\pi i\varphi(2^{t'})} \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(k)} |1\rangle \otimes |k\rangle_{t'} = |1\rangle \otimes (e^{2\pi i\varphi(2^{t'})} \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(k)}|k\rangle_{t'})$.

(7) $\forall_a$, $\forall_b$, $\forall_J$, $\sum_{j\in J} a \otimes b_j = a \otimes \sum_{j\in J} b_j$, $\forall_{t\in\mathbb{N}^+}$, and $\forall_{k\in[2^{t-1}]}$, $|0\rangle \otimes |k\rangle_t = |k\rangle_{t+1}$. Thus $\sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} = \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|0\rangle \otimes |k\rangle_{t'} = |0\rangle \otimes \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|k\rangle_{t'}$.

(8) $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*}=\sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|k\rangle_{t^*}+\sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(k+2^{t'})}|k + 2^{t'}\rangle_{t^*}$, $\sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(2^{t'}+k)}|2^{t'} + k\rangle_{t^*} = |1\rangle \otimes e^{2\pi i\varphi(2^{t'})} \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(k)}|k\rangle_{t'}$, and $\sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k} |k\rangle_{t^*}=|0\rangle \otimes \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|k\rangle_{t'}$. Thus $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} = |0\rangle \otimes \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|k\rangle_{t'} + |1\rangle \otimes e^{2\pi i\varphi(2^{t'})} \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(k)}|k\rangle_{t'}=(|0\rangle+|1\rangle e^{2\pi i\varphi(2^{t'})})\otimes \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(k)}|k\rangle_{t'}$.

(9) $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} = (|0\rangle + |1\rangle e^{2\pi i\varphi(2^{t'})})\otimes \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi(k)}|k\rangle_{t'}$ and $\forall_{k\in]t[}$, $p'_k = (|0\rangle + e^{2\pi i 2^k\varphi}|1\rangle)$ and $\psi'_{t'} = \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|k\rangle_{t'}$. Thus $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} = p'_{t'} \otimes \psi'_{t'}$.

(10) $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} = p'_{t'} \otimes \psi'_{t'}$, $\psi'_1 = p'_0$, and $\forall_{k\in[t-1]}$, $\psi_{k+1} = p'_k \otimes \psi'_k$. Thus $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} = \psi'_{t^*}$.

(11) $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*} = \psi'_{t^*}$ and $\forall_{a,b,\alpha}$, if $a = b$, then $\alpha a = \alpha b$, with $a = \sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*}$, $b = \psi'_{t^*}$, and $\alpha = \frac{1}{2^{t^*/2}}$. Thus $\frac{1}{2^{t^*/2}}\psi'_{t^*} = \frac{1}{2^{t^*/2}} \sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*}$.

(12) $\frac{1}{2^{t^*/2}}\psi'_{t^*} = \frac{1}{2^{t^*/2}} \sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*}$ and $\frac{1}{2^{t^*/2}}\psi'_{t^*} = \psi_{t^*}$. Thus $\psi_{t^*} = \frac{1}{2^{t^*/2}} \sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*}$.

(13) Assuming $\psi_{t'} = \frac{1}{2^{t'/2}} \sum_{k=0}^{2^{t'}-1} e^{2\pi i\varphi k}|k\rangle_{t'}$ and t' is arbitrary, we obtain $\psi_{t^*} = \frac{1}{2^{t^*/2}} \sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}|k\rangle_{t^*}$. Thus, $\forall_{t\in\mathbb{N}^+}, P(t) \Rightarrow P(t + 1)$.

(14) $P(1)$ and $\forall_{t\in\mathbb{N}^+}, P(t) \Rightarrow P(t + 1)$ and the principle of mathematical induction. Thus, $\forall_{t\in\mathbb{N}^+}, P(t)$.

Could we use PROVE-IT's output to generate a structured proof format to convey a proof clearly and concisely? The full formal proof output of `_psi_t_formula`, for example, contains a large number of details that are trivial and distracting to any experts. All of the steps of a good structured proof are contained in the formal proof output, but the excessive details should be stripped away. One idea for doing this would be to present the proof output in an interactively expandable tree structure (much like the way that directories may be navigated in an operating system) and have further options for the user to reorder the steps as desired. Table V illustrates the potential for building a more readily communicable proof sketch from PROVE-IT's output, similar to the proof via dependencies shown above.

[1] IBM Quantum, https://quantum-computing.ibm.com/ (IBM Research, Armonk, 2023).

[2] IBM Q, https://www.rigetti.com/ (Rigetti Computing, Berkeley, 2023).

[3] Oxford Quantum Circuits, https://oxfordquantumcircuits.com/technology/ (Oxford Quantum Circuits, Shinfield, 2023).

[4] Starmon-5, https://www.quantum-inspire.com/backends/starmon-5/ (Quantum Inspire, Delft, 2023).

[5] Trapped Ion Quantum Computing, https://ionq.com/ (IonQ, College Park, 2023).

[6] QSCOUT, https://www.sandia.gov/quantum/quantum-information-sciences/projects/qscout/ (Sandia National Laboratories, Albuquerque, 2023).

[7] Neutral Atom Platform, https://www.quera.com/neutral-atom-platform (QuEra, Boston, 2022).

[8] Photonics, https://xanadu.ai/photonics/ (Xanadu, Toronto, 2023).

[9] Quandela Cloud, https://cloud.quandela.com/ (Quandela, Île-de-France, 2022).

[10] Spin2, https://www.quantum-inspire.com/backends/spin-2/ (Quantum Inspire, Delft, 2023).

[11] P. Shor, in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society, Washington, DC, 1994), pp. 124–134.

[12] D. Boneh and R. J. Lipton, Quantum cryptanalysis of hidden linear functions (Extended Abstract), in *Advances in Cryptology—CRYPTO '95*, edited by D. Coppersmith, Lecture Notes in Computer Science Vol. 963 (Springer, Berlin, 1995), pp. 424–437.

[13] A. W. Harrow, A. Hassidim, and S. Lloyd, Quantum algorithm for linear systems of equations, Phys. Rev. Lett. **103**, 150502 (2009).

[14] M. Motta and J. E. Rice, Emerging quantum computing algorithms for quantum chemistry, WIREs Comput. Mol. Sci. **12**, e1580 (2022).

[15] C. Chareton, S. Bardin, D. Lee, B. Valiron, R. Vilmart, and Z. Xu, Formal methods for quantum programs: A survey, arXiv:2109.06493.

[16] S. Jordan, Quantum Algorithm Zoo, available at https://quantumalgorithmzoo.org/ (2022).

[17] K. Hietala, R. Rand, S.-H. Hung, X. Wu, and M. Hicks, Verified optimization in a quantum intermediate representation, arXiv:1904.06319.

[18] K. Hietala, R. Rand, S.-H. Hung, X. Wu, and M. Hicks, A verified optimizer for quantum circuits, Proc. ACM Program. Lang. **5**, 37 (2021).

[19] J. Paykin, R. Rand, and S. Zdancewic, in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, Paris, 2017* (Association for Computing Machinery, New York, 2017), pp. 846–858.

[20] R. Rand, J. Paykin, and S. Zdancewic, Phantom types for quantum programs, in *The Fourth International Workshop on Coq for Programming Languages* (2018),

https://popl18.sigplan.org/details/CoqPL-2018/13/Phantom-Types-for-Quantum-Programs.

[21] R. Rand, J. Paykin, and S. Zdancewic, QWIRE practice: Formal verification of quantum circuits in Coq, Electron. Proc. Theor. Comput. Sci. **266**, 119 (2018).

[22] Coq Development Team, The Coq proof assistant, https://zenodo.org/record/7313584#.Y36kuOzMKu4 (Zenodo, Geneva, 2022).

[23] The Coq Proof Assistant, https://coq.inria.fr/.

[24] C. Chareton, S. Bardin, F. Bobot, V. Perrelle, and B. Valiron, in *Programming Languages and Systems: 30th European Symposium on Programming*, edited by N. Yoshida, Lecture Notes in Computer Science Vol. 12648 (Springer, Cham, 2021), pp. 148–177.

[25] Qbricks, Qbricks, https://qbricks.github.io/.

[26] T. Liu, Y. Li, S. Wang, M. Ying, and N. Zhan, A theorem prover for quantum Hoare logic and its applications, arXiv:1601.03835.

[27] J. Liu, B. Zhan, S. Wang, S. Ying, T. Liu, Y. Li, M. Ying, and N. Zhan, in *Computer Aided Verification*, edited by I. Dillig and S. Tasiran (Springer, Cham, 2019), pp. 187–207.

[28] J. Liu, B. Zhan, S. Wang, S. Ying, T. Liu, Y. Li, M. Ying, and N. Zhan, Quantum Hoare logic, Archive of Formal Proofs, https://isa-afp.org/entries/QHLProver.html (2019).

[29] M. Ying, Floyd-Hoare logic for quantum programs, ACM Trans. Program. Lang. Syst. **33**, 19 (2011).

[30] M. Ying, Toward automatic verification of quantum programs, Form. Asp. Comput. **31**, 3 (2018).

[31] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th Anniversary Edition (Cambridge University Press, New York, NY, 2010).

[32] W. M. Witzel, W. D. Craft, R. D. Carr, and J. E. M. Larrañaga, Prove-It: A proof assistant for organizing and verifying general mathematical knowledge, arXiv:2012.10987.

[33] Prove-It, Welcome to the expanding Prove-It library of proofs, www.pyproveit.org (Sandia National Laboratories, Albuquerque, 2023).

[34] Prove-It, Prove-It GitHub development site, https://github.com/sandialabs/Prove-It (Sandia National Laboratories, Albuquerque, 2023).

[35] J. R. Shoenfield, *Mathematical Logic* (Addison-Wesley, Reading, 1967).

[36] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, Quantum algorithms revisited, Proc. Math. Phys. Eng. Sci. **454**, 339 (1998).

[37] Prove-It, Welcome to the expanding Prove-It library of proofs, https://pyproveit.github.io/Prove-It/2023_03_20/ (Sandia National Laboratories, Albuquerque, 2023).

[38] N. D. Mermin, *Quantum Computer Science: An Introduction.* (Cambridge University Press, Cambridge, 2007), p. 5.

[39] G. Benenti, G. Casati, and G. Strini, *Principles of Quantum Computation and Information* (World Scientific, Singapore, 2004), Vol. 1.

[40] K. Hietala, R. Rand, S.-H. Hung, L. Li, and M. Hicks, in *12th International Conference on Interactive Theorem Proving (ITP 2021)*, edited by L. Cohen and C. Kaliszyk, Leibniz International Proceedings in Informatics (LIPIcs) Vol. 193 (Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, 2021), pp. 21:1–21:19.

[41] R. E. Hodel, *An Introduction to Mathematical Logic* (Dover, Mineola, 1995).

[42] E. Mendelson, *Introduction to Mathematical Logic*, 6th ed. (CRC, Boca Raton, 2015).

[43] P. Martin-Löf, On the meaning of the logical constants and the justifications of the logical laws, Nord. J. Philos. Logic **1**, 11 (1996).

[44] L. Lamport, How to write a proof, Am. Math. Mon. **102** (7), 600 (1995).

[45] L. Lamport, How to write a 21st century proof, J. Fixed Point Theory Appl. **11**, 43 (2012).

[46] H. Barendregt and E. Barendsen, Autarkic computations in formal proofs, J. Autom. Reasoning **28**, 321 (2002).

[47] H. Barendregt and W. Freek, The challenge of computer mathematics, Philos. Trans. R. Soc. A **363**, 2351 (2002).

[48] B. Eastin and S. T. Flammia, Q-circuit tutorial, arXiv:quant-ph/0406003.

[49] N. G. de Bruijn, in *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, edited by J. Seldin and J. Hindley (Academic, New York, 1980), pp. 579–606.

[50] F. Wiedijk, The De Bruijn factor, available at https://www.cs.ru.nl/~freek/factor/factor.pdf (unpublished).

[51] F. Bauer-Marquart, S. Leue, and C. Schilling, symQV: Automated symbolic verification of quantum programs, in *Formal Methods. FM 2023*, Lecture Notes in Computer Science, Vol. 14000, edited by M. Chechik, J. P. Katoen, and M. Leucker (Springer, Cham, 2023), pp. 181–198.

[52] M. Amy, Formal methods in quantum circuit design, Ph.D. thesis, University of Waterloo, 2019, available at https://uwspace.uwaterloo.ca/bitstream/handle/10012/14480/Amy_Matthew.pdf.

[53] C. Lattner and V. Adve, in *Proceedings of the International Symposium on Code Generation and Optimization, San Jose, 2004* (IEEE, Piscataway, 2004).

[54] The LLVM Compiler Infrastructure project, llvm.org (LLVM Foundation, Los Altos).

[55] Why3, https://why3.lri.fr/ (Inria Saclay/Université Paris-Saclay/CNRS, Île-de-France, 2023).

[56] W. Witzel, M. Sarovar, and K. Rudinger, Versatile formal methods applied to quantum information, prod.sandia.gov/techlib/access-control.cgi/2015/159617r.pdf (Sandia National Laboratories, Albuquerque, 2015).