使用 Prove-It 验证量子相位估计（QPE）

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, NM 87123*

[2] *Department of Computer Science, The University of New Mexico, Albuquerque, NM 87131*

(Dated: April 4, 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.

arXiv:2304.02183v1 [quant-ph] 5 Apr 2023

I. 引言与动机

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/allocation such as IBM, Amazon, Intel, Google, Facebook, *etc*). 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].

与经典计算算法及其在经典计算机上的实现一样，量子算法及其实现的可靠性和正确性至关重要。量子电路可能很复杂，而且（往往）有悖直觉。与经典计算类似，-

sical software, and 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.

A. 验证量子算法和量子电路

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 [16, 17]; QWIRE [18–20], a "quantum circuit language" embedded in the Coq proof assistant [21, 22], allowing users to write quantum circuits using high-level abstractions and to prove properties of those circuits using Coq's theorem-proving features; Qbricks [23, 24], an environment for automated formal verification of quantum programs, taking advantage of the characterization of quantum circuits

as parameterized path-sums; and Quantum Hoare Logic (QHL) [25–29], an extension of Hoare logic for reasoning about quantum programs.

B. Prove-It 定理证明助手

This paper reports on a verification of the quantum phase estimation (QPE) algorithm [30, pp 221–226] using a newly developed theorem-proving assistant called **Prove-It** [31, 32]. 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 [30] 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 quick example of this process appears in Fig. 1, showing the interactive steps (left) and eventual **Prove-It**-generated formal proof (right). This process is discussed in more detail in §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 `http://pyproveit.org`. The system is available for download and user contributions at `https://github.com/PyProveIt/Prove-It`. 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 not-yet-proven theorems and axioms required by the formal proof as well as a list of the theorems that directly rely on that particular theorem in their proofs.

C. 论文的组织结构

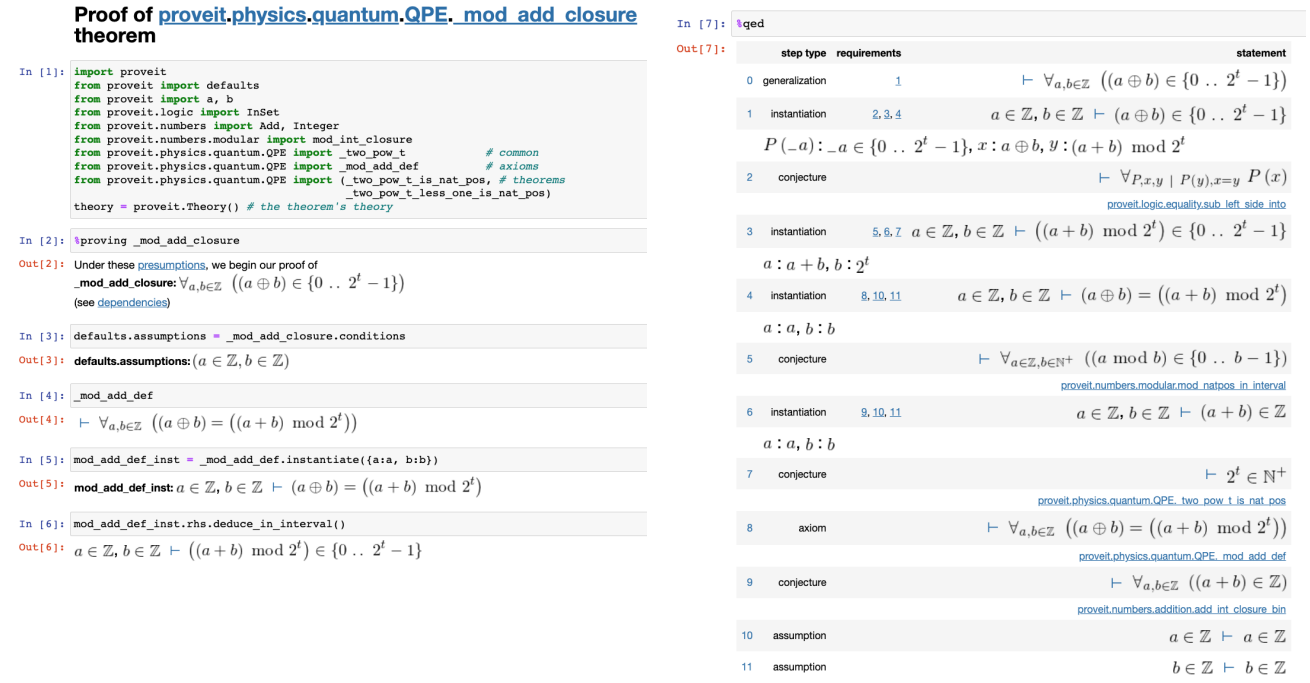
The next three sections of this paper will address the following questions: What, essentially, did we prove (§II)? What, precisely, did we prove (§III)? and How did we construct our proofs (§IV)? These topics are brought together in a demonstration of one specific proof within the QPE package in §V.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

§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; (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. §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.

§III drills down to discuss more precisely just what we've proven. We explain how **Prove-It** traces the proven lemmas/theorems back to the axioms and yet-to-be-proven theorems ("conjectures") on which they depend, and how **Prove-It** avoids circular dependencies. We also provide specific information about the lemmas/theorems that are proven in our QPE theory package and correspond to the named statements of §II. We list other theory packages that our proofs rely upon and the precise number of axioms and "conjectures" (general, fundamental facts that simply have not been proven in our system yet) that were used. We expect that many of the statements that we call axioms at this time will be transformed into conservative definitions, 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.

§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 QPE-specific lemmas/theorems (corresponding to named statements of §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 §V to demonstrate **Prove-It**'s features in one concrete example. Then, in §VI, we make specific com-



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: ⊢ ∀_{a,b∈ℤ} ((a ⊕ b) ∈ {0 .. 2^t − 1})
        (see dependencies)

In [3]: defaults.assumptions = _mod_add_closure.conditions
Out[3]: defaults.assumptions: (a ∈ ℤ, b ∈ ℤ)

In [4]: _mod_add_def
Out[4]: ⊢ ∀_{a,b∈ℤ} ((a ⊕ b) = ((a + b) mod 2^t))

In [5]: mod_add_def_inst = _mod_add_def.instantiate({a:a, b:b})
Out[5]: mod_add_def_inst: a ∈ ℤ, b ∈ ℤ ⊢ (a ⊕ b) = ((a + b) mod 2^t)

In [6]: mod_add_def_inst.rhs.deduce_in_interval()
Out[6]: a ∈ ℤ, b ∈ ℤ ⊢ ((a + b) mod 2^t) ∈ {0 .. 2^t − 1}

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

	step type	requirements	statement
0	generalization	1	$\vdash \forall_{a,b\in\mathbb{Z}}\ ((a\oplus b)\in\{0\,..\,2^t-1\})$
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	$a\in\mathbb{Z}, b\in\mathbb{Z} \vdash ((a+b) \bmod 2^t)\in\{0\,..\,2^t-1\}$
			$a: a+b, b: 2^t$
4	instantiation	8, 10, 11	$a\in\mathbb{Z}, b\in\mathbb{Z} \vdash (a\oplus b)=((a+b) \bmod 2^t)$
			$a: a, b: b$
5	conjecture		$\vdash \forall_{a\in\mathbb{Z},b\in\mathbb{N}^+}\ ((a \bmod b)\in\{0\,..\,b-1\})$
			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		$\vdash \forall_{a,b\in\mathbb{Z}}\ ((a\oplus b)=((a+b) \bmod 2^t))$
			proveit.physics.quantum.QPE._mod_add_def
9	conjecture		$\vdash \forall_{a,b\in\mathbb{Z}}\ ((a+b)\in\mathbb{Z})$
			proveit.numbers.addition.add_int_closure_bin
10	assumption		$a\in\mathbb{Z} \vdash a\in\mathbb{Z}$
11	assumption		$b\in\mathbb{Z} \vdash b\in\mathbb{Z}$

图 1： `mod add closure`(局部 QPE) 定理证明笔记本的摘录，展示交互式步骤（左侧）以及**Prove-It**生成的形式化证明（右侧）。

parisons 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 §VII and acknowledgments in §VIII.

II. 我们本质上证明了什么?

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 the website at `pyproveit.org`. 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 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 & Chuang's [30] standard textbook development and eventual proof of three main claims about the QPE outcome probabilities (also see [33]). 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 §IV.

在量子相位估计（QPE）中，我们有一个幺正算子

矩阵 U 使得

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

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. 2. 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 eigenvector $|u\rangle$. A Hadamard is applied to each line in the first register, 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 $\mathrm{QFT}^\dagger$ 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. 2, 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

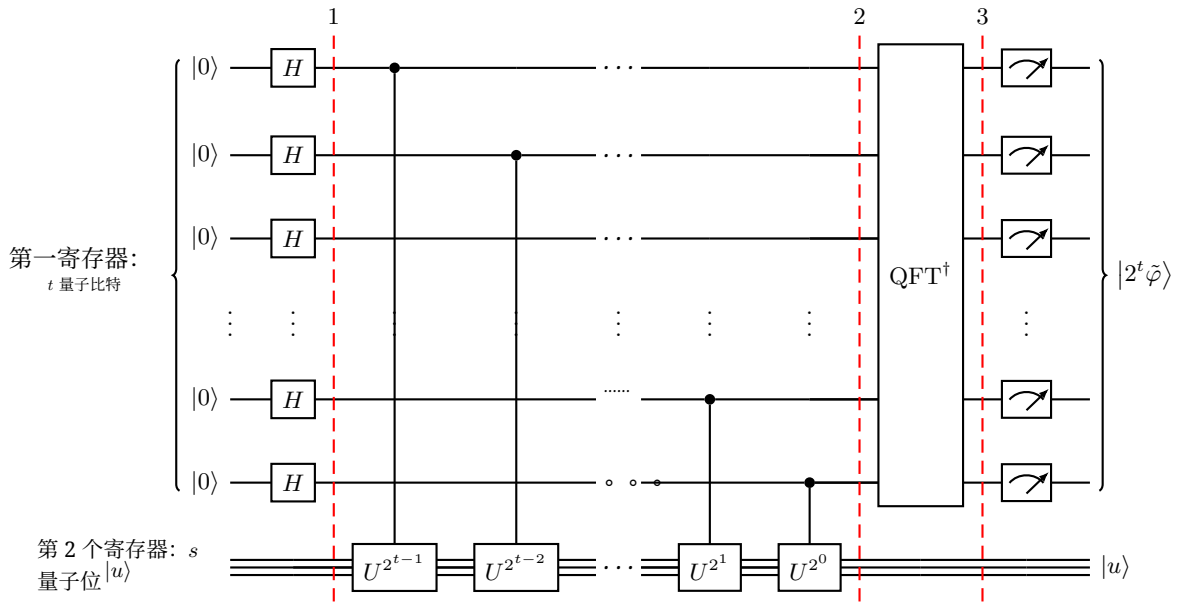


FIG. 2: Quantum circuit implementing the QPE algorithm. From initial first-register state $|0\rangle_t$ and second register state $|u\rangle$, we apply Hadamards to the first register lines, controlled U^{2^j} to the second register, and an inverse quantum Fourier transform $\mathrm{QFT}^\dagger$ to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

假设包含在以下内容中:

QPE 假设: t 属于正自然数 $t \in \mathbb{N}^+$ (2)

QPE Assumption: `_s_in_nat_pos` $s \in \mathbb{N}^+$ (3)

QPE Assumption: `_unitary_U` $U \in \mathrm{U}(2^s)$ (4)

QPE Assumption: `_u_ket_register` $|u\rangle \in \mathbb{C}^{2^s}$ (5)

QPE Assumption: `_normalized_ket_u` $||\,|u\rangle\,|| = 1$ (6)

QPE Assumption: `_phase_in_interval` $\varphi \in [0, 1)$ (7)

QPE Assumption: `_eigen_uu` $U\,|u\rangle = e^{2\pi i\varphi}\,|u\rangle$ (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).

我们还定义了一种特定的模加法，它将在若干引理的陈述中用到:

Local QPE Definition: `_mod_add_def`

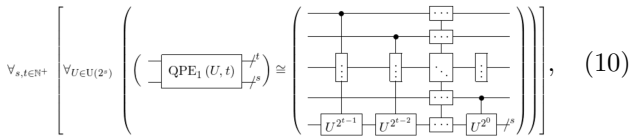
$$\forall_{a,b\in\mathbb{Z}}((a \oplus b) = ((a + b) \bmod 2^t)), \quad (9)$$

其中 $t \in \mathbb{N}^+$ 同样是第一个寄存器中可用的量子比特数。该定义是局部的，因为它是针对 t 的问题特定值定义的。

为方便起见，我们定义了 QPE 算法电路中所示各部分的一些简化表示。

图 2. 我们将切片 1 和 2 之间的电路部分表示为门 $\mathrm{QPE}_1(U, t)$，其定义如下:

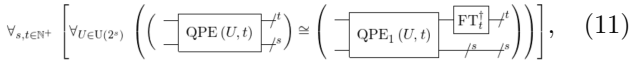
QPE 定义: QPE1 定义



(10)

以及切片 1 和 3 之间的电路部分作为门 $\mathrm{QPE}(U, t)$，定义为:

QPE 定义: QPE def_



(11)

where the symbol ≅ denotes circuit equivalence. Notice that the circuit equivalence `QPE_def` (11) utilizes $\mathrm{QPE}_1(U, t)$ 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 initial first-register state $|0\rangle_t$ with the subsequent Hadamards to consider the t-qubit state $|+\rangle\otimes\ldots\otimes|+\rangle$ 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. 阶段 2 的 QPE 电路状态

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

局部 QPE 定理：psi t 输出_ _

$$\forall_{t\in\mathbb{N}^+}\left(\Pr\left(\begin{array}{l}|+\rangle\\ |+\rangle\\ \vdots\\ (t-3)\times\ \mathrm{QPE}_1\,(U,t)\\ \vdots\\ |+\rangle\\ |u\rangle\end{array}\begin{array}{l}\\ \\ \\ \\ \\ \\ \end{array}\begin{array}{l}\\ \\ \\ |\psi_t\rangle\\ \\ \\ |u\rangle\end{array}\right)=1\right),\tag{12}$$

其中 $|\psi_t\rangle$ 在（局部）定义为

局部 QPE 定义：psi t def _ _

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

Although _psi_t_def (13) quantifies over t as a variable, it is local because it relies upon a problem-specific φ. _psi_t_output (12) is also local, implicitly relying upon problem-specific assumptions and definitions. _psi_t_output combined with _psi_t_def correspond to the right-hand side of Nielsen & Chuang's circuit in their Fig. 5.2, and the left-hand side of their equation (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.

ψ t 输出 (12) 的证明依赖于更基本的"相位回踢"现象，这一现象由**Prove-It**的量子电路包定理表述为:

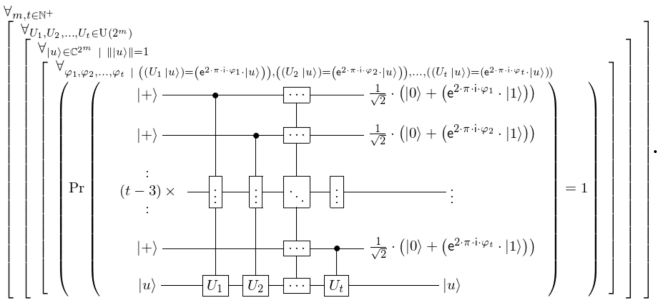
quantum.circuit Theorem: phase_kickback

$$\forall_{m\in\mathbb{N}^+}\left[\forall_{U\in\mathrm{U}(2^m)}\left[\forall_{|u\rangle\in\mathbb{C}^{2^m}\ |\ \||u\rangle\|=1}\left[\forall_{\varphi\ |\ (U\,|u\rangle)=(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi}\cdot|u\rangle)}\right.\right.\right. \\ \left.\left.\left.\left(\Pr\left(\begin{array}{ll}|+\rangle & \frac{1}{\sqrt{2}}\cdot(|0\rangle+(\mathrm{e}^{2\cdot\pi\cdot\mathrm{i}\cdot\varphi}\cdot|1\rangle))\\ |u\rangle & |u\rangle\end{array}\right)=1\right)\right]\right]\right],\tag{14}$$

并推广至多个量子寄存器上的应用。

量子比特作
为

quantum.circuit 定理:
寄存器上的相位回踢 _



(15)

然后，我们证明 psi t def (13) 中的张量积可以改写为与Nielsen & Chuang 的公式(5.20) 相对应的紧凑求和形式 [30, pg 222]:[1]

Local QPE Theorem: _psi_t_formula

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

The _psi_t_formula theorem 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 $t=2$ shown below:

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

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

[1] $|k\rangle_t$ denotes a *number ket* (a `NumKet` object in **Prove-It**), similar to notation used in Mermin [34, pg 5] for a multi-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 5 qubits, for example, can be represented by the number ket $|26\rangle_5$.

B. 第 3 阶段的 QPE 电路状态

接下来，我们证明第 3 阶段的第一个寄存器状态为$|\Psi\rangle$:

局部 QPE 定理：Psi 输出 _

$$\Pr\left(\begin{array}{c}\text{QPE}\,(U,t)\\ \text{with}\\ |\Psi\rangle\end{array}\right)=1\,, \tag{19}$$

其中，$|\Psi\rangle$ 在本地由 Psi def 定义为：_

$$\begin{array}{l}\text{Local QPE Definition: _Psi_def}\\ |\Psi\rangle=\mathrm{FT}_t^{\dagger}\,|\psi_t\rangle\,,\end{array} \tag{20}$$

并且 $\mathrm{FT}_t^{\dagger}$ 表示一个 t量子比特的逆量子傅里叶变换。这显然可以由QPE 电路的定义（定义 11）直接得出。当然，逆量子傅里叶变换本身也是一种重要的量子算法，但在此处将其视为一个模块化的“黑盒”函数，该函数返回一个定义明确的结果。

与 psi t 输出 _(12）一样，Psi 输出（19）表示为一个概率——即在第 3 阶段，从给定输入中获得第一寄存器状态 $|\Psi\rangle$ 和第二寄存器状态 $|u\rangle$ 的概率为 1。

C. 测量结果的概率空间

为严谨地形式化 QPE 过程及其相关测量概率，我们谨慎地建立一个概率空间，以形式化地建模 QPE 电路的测量结果。

Interpreting the sequence of 0s and 1s 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 $|+\rangle\otimes|+\rangle\otimes\ldots\otimes|+\rangle\otimes|u\rangle$ and (probabilistic) first-register measurement result $|m\rangle_t$, with $m\in\{0,1,\ldots 2^t-1\}$:

LocalQPEDefinition: 样本空间定义 _

$$\Omega=\left\{\begin{array}{c}|+\rangle\\ |+\rangle\\ \vdots\\ (t-3)\times\\ \vdots\\ |+\rangle\\ |u\rangle\end{array}\right.\!\!\begin{array}{c}\\ \\ \\ \text{QPE}\,(U,t)\\ \\ \\ \\ \end{array}\!\!\left.\begin{array}{c}\\ \\ \\ |m\rangle_t\\ \\ \\ |u\rangle\end{array}\right\}_{m\in\{0\,..\,2^t-1\}} \tag{21}$$

其中，样本空间定义（21）的右侧是 集合构造表达式在 **Prove-It** 中的一个示例 —— 即，样本空间定义的右侧表示一个由输入—输出—电路配置组成的集合，而大的花括号是集合定界符。[2]

要证明 Ω具有样本空间的性质，我们需要确立从整数 m到带有测量 $|m\rangle_t$ 的QPE 电路配置的映射，是从集合 $\{0,1,2,\ldots,2^t-1\}$ 到 Ω 上的一一映射：

LocalQPETheorem：样本空间双射 _

$$\left[m\mapsto\left(\begin{array}{c}|+\rangle\\ |+\rangle\\ \vdots\\ (t-3)\times\\ \vdots\\ |+\rangle\\ |u\rangle\end{array}\right.\!\!\begin{array}{c}\\ \\ \\ \text{QPE}\,(U,t)\\ \\ \\ \\ \end{array}\!\!\left.\begin{array}{c}\\ \\ \\ |m\rangle_t\\ \\ \\ |u\rangle\end{array}\right)\right]\in\left[\{0\,..\,2^t-1\}\ \overset{\text{1-to-1}}{\underset{\text{onto}}{\rightarrow}}\ \Omega\right] \tag{22}$$

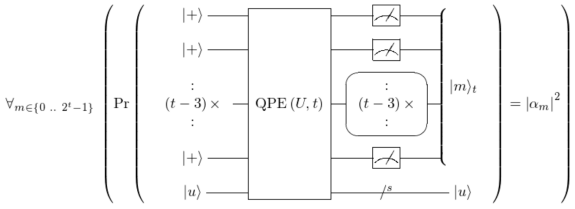
This means that the values $m\in\{0,1,2,\ldots,2^t-1\}$ induce a partition of the sample space Ω, which is true given that quantum circuits with distinct outputs are distinct from each other. This is also critical later in evaluating the probability of an *event* consisting of the disjunction $\{|m_1\rangle_t,|m_2\rangle_t,\ldots,|m_k\rangle_t\}$ 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 measure-

[2]Prove-It 中的集合构造表达式看起来与标准的集合构造表达式非常相似，只是定义域说明以脚标形式出现在末尾。例如，Prove-It 中用于正实数的一个集合（子集）构造表达式可能写作 $\{x\,|\,x>0\}_{x\in\mathbb{R}}$。

ment):

Local QPE Theorem: _outcome_prob



(23)

其中 α_m 是与测量结果 $|m\rangle$ 对应的概率幅：

Local QPE Definition: _alpha_m_def

$$\forall_{m\in\{0,\ldots,2^t-1\}}\big(\alpha_m = \langle m|\Psi\rangle\big). \tag{24}$$

然后我们可以证明，样本空间定义 (21) 中定义的集合 Ω 确实是一个样本空间：

Local QPE Theorem:_Omega_is_sample_space

$$\Omega \underset{c}{\in} \text{SampleSpaces} \tag{25}$$

这意味着 Ω 中指定的结果互斥且穷尽，因此也意味着 Ω 中各个结果的概率之和为 1。[3]

D. 评估概率幅度 α_m

然后我们证明 α_m 的一个初始公式。正如 Nielsen & Chuang [30, 第 223 页的公式 (5.23)] 所指出的，对 psi t 公式 (16) 中的态应用逆傅里叶变换，会得到如下双重求和态：

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

根据 alpha m def (24) 中 α_m 的定义和 psi t formula (16)，我们可以推导出与测量结果 $|m\rangle$ 对应的概率振幅公式：

局部 QPETheorem：alpha m 评估

$$\forall_{m\in\{0,\ldots,2^t-1\}}\ \left(\alpha_m = \left(\frac{1}{2^t}\sum_{k=0}^{2^t-1}\left(e^{-\frac{2\pi i k m}{2^t}}\cdot e^{2\pi i\varphi k}\right)\right)\right). \tag{27}$$

[3]带有“underset”的 $\in$ 记号 c 用于表示类成员关系，并将其与标准的集合成员关系区分开来。这里我们只是把所有样本空间的汇集视为一个类，这比假定该汇集实际上构成一个集合更弱（只有集合才允许子集概括）。

这直接由 $|\Psi\rangle$(13)、$|\psi_t\rangle$ (20) 的定义得出，并且

QFT 定理：矩阵元素上的_invFT _

$$\forall_{n\in\mathbb{N}^+}\ \Big[\forall_{k,l\in\{0\ ..\ 2^n-1\}} \left(\left({}_n\langle l|\,\mathrm{FT}_n^\dagger\,|k\rangle_n\right) = \left(\frac{1}{2^{\frac{n}{2}}}\cdot \mathrm{e}^{\frac{-(2\cdot\pi\cdot\mathrm{i}\cdot k\cdot l)}{2^n}}\right)\right)\Big]. \tag{28}$$

_ alpha_m 求值 (27) 明确了 Nielsen & Chuang 中一个隐含步骤，该步骤介于他们的表达式 (5.23) 与公式 (5.24) [30, pg 223] 之间。

E. 特殊情况：相位 φ 可用 t 位精确表示

对于这一特殊情况，若图 2 中 QPE 电路的第一个含 t 个量子比特的寄存器足以精确表示相位 φ（Nielsen 和 Chuang[30, 第 223] 页称之为“理想情况”），则“理想相位条件”为：

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

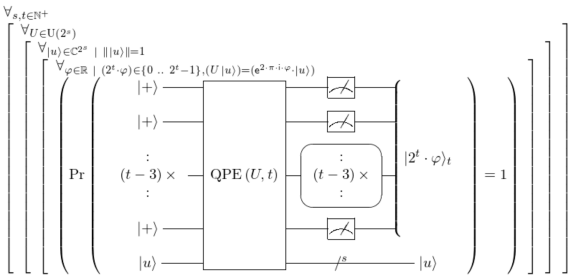
在此条件下，用 $m = 2^t\varphi$ 实例化 alpha m 评估式 (32) 会导致

LocalQPETheorem: alpha 理想情况

$$\big((2^t\varphi) \in \{0, 1, 2, \ldots, 2^t - 1\}\big) \Rightarrow \big(\alpha_{(2^t\varphi)} = 1\big) \tag{30}$$

也就是说，缩放后的相位 $2^t\varphi$ 的概率振幅为 1。使用结果概率 (23) 和理想情形下的 alpha (30),

QPE 定理：qpe exact



(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. 界定最佳测量结果的概率 b_r =round$(2^t\varphi)$

当 $2^t\varphi$ 无法在第一个寄存器中可用的 t 个量子比特上精确表示时，我们仍然可以证明一个下界

获得最佳可能测量结果的概率上界， b_r =round($2^t\varphi$)。

根据 alpha_m 的评估 (27)，并利用 2^t 对 $e^{2\pi ikm/2^t}$ 的周期性，我们可以推导出

LocalQPE 定理：alpha m 模运算求值

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

然后将其明确改写为有限几何级数：

LocalQPETheorem：alpha m_mod 作为几何级数_

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

现在我们将 b_r 和 δ_b 定义如下：

Local QPE Definition: _best_round_def

$$b_r = \texttt{round}(2^t\varphi); \tag{34}$$

局部 QPE 定义：delta b 定义 _ _

$$\forall_{b\in\mathbb{Z}}\left(\delta_b = \varphi - \frac{b}{2^t}\right). \tag{35}$$

将有限几何和 alpha m mod 化简为几何和 (33)，并使用$\theta > \theta \Rightarrow \sin\theta \prec \theta$和 $0 \leq \theta \leq \pi/2 \Rightarrow \sin\ \theta \geq 2\theta/\pi$（如图 3 所示），可得

局部 QPE 定理：最佳保证为 delta 非零_ _

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

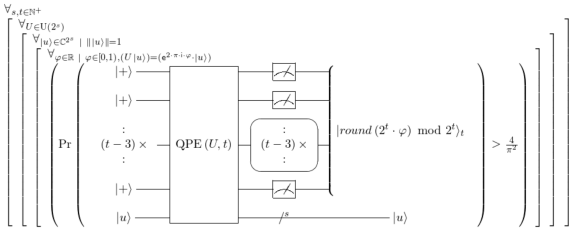
That is: when the t qubits of the first register are insufficient to exactly represent the scaled phase $2^t\varphi$, *i.e.*, $(\delta_{b_r} \neq 0)$, the squared norm of the probability amplitude of the rounding-based best estimate b_r is greater than $4/\pi^2$. Of course, this doesn't really require that $\delta_{b_r} \neq 0$ since qpe_exact (31) proves that the probability is 1 when $\delta_{b_r} = 0$. Therefore,

Local QPE Theorem: _best_guarantee

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

进而导出一般的 QPE 最优保证定理，给出了_ob- 的概率下界得到 b_r =round($2^t\varphi$) 作为测量结果：

QPE 定理：QPE 的最佳保证



, (38)

这与此前的非形式化结果 [33, 35]以及最近在 *Coq* [36], 中得出的形式化证明结果一致，不过他们使用的是较弱的（非严格）不等式（$\geq$）。

G. 保证所需精度

Closely matching Nielsen & Chuang's [30, pg 223] approach in their §5.2.1 on "Performance and Requirements," let the integer $b_f \in \{0, 1, 2, \ldots, 2^t - 1\}$ be such that $b_f/2^t = 0.b_1b_2\ldots b_t$ 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

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

产生差值 $\delta_{b_f} = \varphi - \frac{b_f}{2^t} > 0$；回顾 δ (35) 的一个较早定义，并注意到 §II E 中 $b_f = b_r = \varphi$ 产生 $\delta_{b_f} = 0$ 的"理想情况"已经考虑过。

根据几何级数（33）中关于 alpha m mod 的有限几-何和表达式，-我们可以证明 -

Local QPE Theorem: _alpha_summed

$$\forall_{\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\,|\,\ell\neq 0} \left(\alpha_{b_f\oplus\ell} = \left(\frac{1}{2^t}\cdot\frac{1-\mathrm{e}^{2\pi\mathrm{i}\left(2^t\delta_{b_f}-\ell\right)}}{1-\mathrm{e}^{2\pi\mathrm{i}\left(\delta_{b_f}-\frac{\ell}{2^t}\right)}}\right)\right), \tag{40}$$

which corresponds to expression (5.26) in Nielsen & Chuang [30, pg 224] with minor differences in notation and the domain for ℓ explicitly stated. _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 & Chuang's expression (5.24):

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

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

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

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 §II E.

令 e 为"表征我们所需误差容限的正整数" [30, pg 224], , 我们用以下定义来规定测量结果"成功"的概率（及其互补事件"失败"的概率）：

局部 QPE 定义：成功定义 _

$$\forall_{e \in \{1\,..\,2^{t-1}-2\}} \left([P_{\text{success}}](e) = \left[\text{Prob}_{m \in \{0\,..\,2^t-1\} \mid |m-b_f|_{\text{mod}\,2^t} \le e} \left(\begin{array}{l} |+\rangle \\ |+\rangle \\ \vdots\ (t-3)\times \\ |+\rangle \\ |u\rangle \end{array} \right) \right] \right), \tag{43}$$

局部 QPE 定义：失败定义 _

$$\forall_{e \in \{1\,..\,2^{t-1}-2\}} \left([P_{\text{fail}}](e) = \left[\text{Prob}_{m \in \{0\,..\,2^t-1\} \mid |m-b_f|_{\text{mod}\,2^t} > e} \left(\begin{array}{l} |+\rangle \\ |+\rangle \\ \vdots\ (t-3)\times \\ |+\rangle \\ |u\rangle \end{array} \right) \right] \right). \tag{44}$$

_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 under-estimate b_f of $2^t\varphi$ is simply the probability of obtaining the QPE circuit configuration with first-register measurement output $|m\rangle_t$ such that $|m - b_f|_{\text{mod}\,2^t} \le e$. 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 & Chuang express the failure probability using the notation $p(|m - b| > e)$ [30, pg 224, lhs of Eq (5.27)]. Although the constraint is unstated in Nielsen & 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.

回顾之前的内容，$\alpha_{b_f \oplus \ell}$ 是向量 $|b_f \oplus \ell\rangle_t$的概率幅值，其中 b_f 是缩放相位 $2^t\varphi$的最佳 t位低估值。我们还定义了 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 _fail_sum (45):

LocalQPETheorem：失败求和

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

which corresponds to formula (5.27) in Nielsen & Chuang [30, pg 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 $m - b_f$):

局部 QPE 定理：失败概率总和条件的等价引理 _

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

由 alpha 求和式 (40)，我们可以据此得出 $|\alpha_{b_f \oplus \ell}|^2$ 的上界：

LocalQPE 定理：alpha 平方上界 _ _

$$\forall_{l \in \{-2^{t-1}+1, \ldots, 2^{t-1}\}} \left(\left|\alpha_{b_f \oplus \ell}\right|^2 \le \frac{1}{4\left(\ell - \left(2^t \delta_{b_f}\right)\right)^2} \right), \tag{47}$$

对应于Nielsen & Chuang 中的公式(5.29) [30,第 224] 页，并利用 $\frac{2}{\pi}\theta \le |1 - e^{i\theta}| \le 2$对于 $\theta \in [0, \pi]$ 的界（如图 3 所示）。

由 alpha sqrd less (47)，我们进而推导出 $[P_{\text{fail}}](e)$ 的初始上界表达式：

LocalQPE 定理：失败概率上界引理 _ _

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

对应 Nielsen & Chuang [30, pg 224] 中的界限 (5.30)。应用一些标准

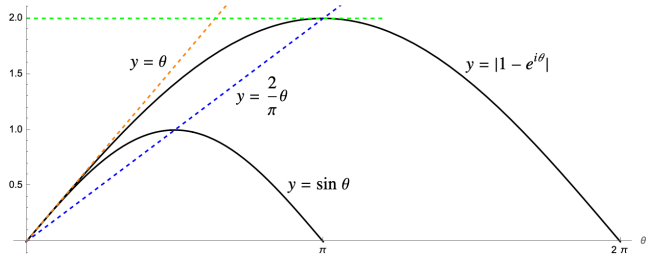


FIG. 3: Illustration of the bounds $\frac{2}{\pi}\theta \leq |1 - e^{i\theta}| \leq 2$ for $\theta \in [0, \pi]$ and $\sin\theta \geq \frac{2}{\pi}\theta$ for $\theta \in [0, \frac{\pi}{2}]$, utilized in the proof of `_alpha_sqrd_upper_bound` (47), and the bound $\sin\theta < \theta$ for $\theta > 0$ used in the proof of `_best_guarantee_delta_nonzero` (36).

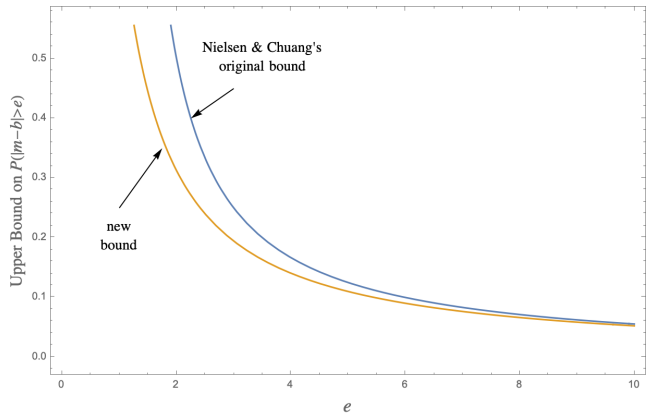


图 4：Nielsen 与 Chuang 的 [30] 原始上界与我们提出的更紧上界的比较，展示了“失败”概率随相对于“最佳”基于下取整的估计值 e 的单步操作数 b_f 变化的情况。

将求和变形和不等式应用于失败上界引理（48），然后我们证明 - - -

局部 QPE 定理：失败上界 - -

$$\forall_{e\in\{1\,..\,2^{t-1}-2\}} \left([P_{\text{fail}}](e) \leq \frac{1}{2e} + \frac{1}{4e^2} \right), \tag{49}$$

corresponding to bound (5.34) in Nielsen & Chuang [30, pg 224], but actually a slight improvement over their bound in two different ways: first, the bound holds for $e \geq 1$ whereas their original bound holds only for $e \geq 2$; and second, the bound in `_failure_upper_bound` (49) is actually tighter (*i.e. less*) than Nielsen & Chuang's bound, and thus the eventually-derived probability for a "successful" measurement is actually *larger*. See Fig. 4 for a graphical comparison of the original and new bounds.

根据成功定义(43)和失败定义(44)，我们可以轻易推导出

局部 QPE 定理：成功与失败互为补集 -

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

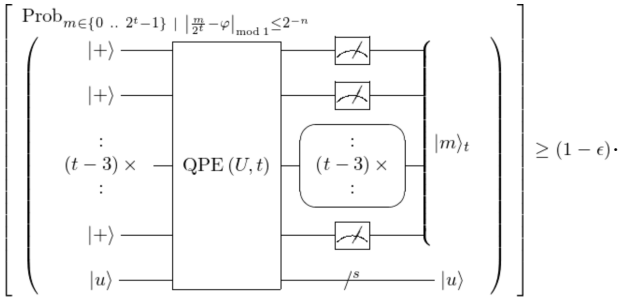
然后我们可以证明，采用 Nielsen & Chuang 的公式 (5.35) [30, pg 224] 作为假设的下界

关于第一个寄存器中的量子比特数量：

$$\text{QPE Assumption: } \texttt{_t_req}$$
$$t \geq \left(n + \left\lceil \log_2 \left(2 + \frac{1}{2\epsilon} \right) \right\rceil \right) \tag{51}$$

足以保证成功测量相位 φ，精度达到 n 位的概率至少为 $1 - \epsilon$：

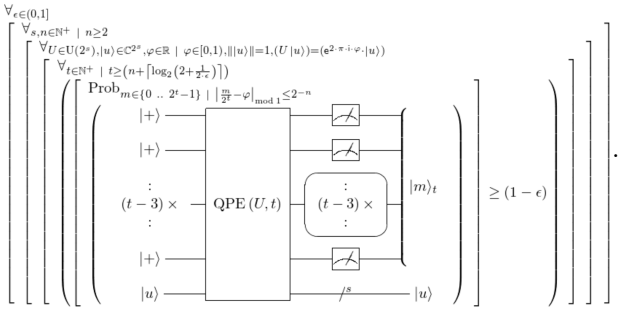
LocalQPETheorem: 精度保证 -



(52)

随后，该局部定理被推广（例如 对所有失败概率 $\epsilon \in (0, 1]$ 以及所有满足原始 QPE 问题陈述的条件）进行推广），从而得出普适的 QPE 精度保证定理：

QPE 定理：qpe 精度保证 -



(53)

III. 我们究竟证明了什么?

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, and an outline of the steps leading to their derivation, are shown in §II. 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. Prove-It 中的形式化证明

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 [37, 38]. 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 rules of inference include modus ponens, deduction, instantiation, and generalization (described in detail in Appendix C of [31]). A formal proof in **Prove-It** also includes several helpful characteristics of a "structured proof" [39, 40], 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. 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 [32], 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 non-traditional 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 appears in Fig. 5a. The steps are numbered in blue 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.

除非另有说明，我们提及证明步骤的顺序时，均指其逆序。

Each step explicitly names the type of derivation step being taken and the previous (higher-numbered) steps required for the step. In Fig. 5a, 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 $\vdash$ 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 结论，如图5b所示，对于步骤2生成的"子证明"。

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. As future work, we plan to develop a lightweight, independent checker that is also open-source with detailed documentation to garner more trust. 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 [41, 42].

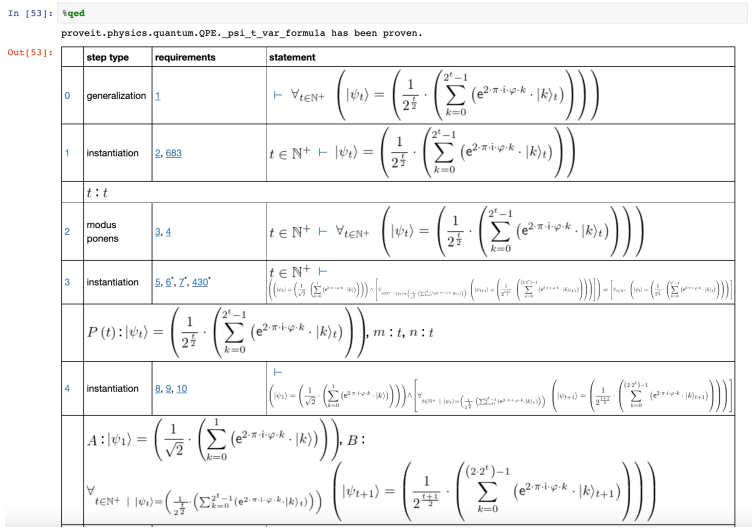
B. 定理依赖关系

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 [32] by navigating to the theorems page of `proveit.physics.quantum.QPE` package, clicking on the corresponding theorem name, and then clicking on the "dependencies" link.

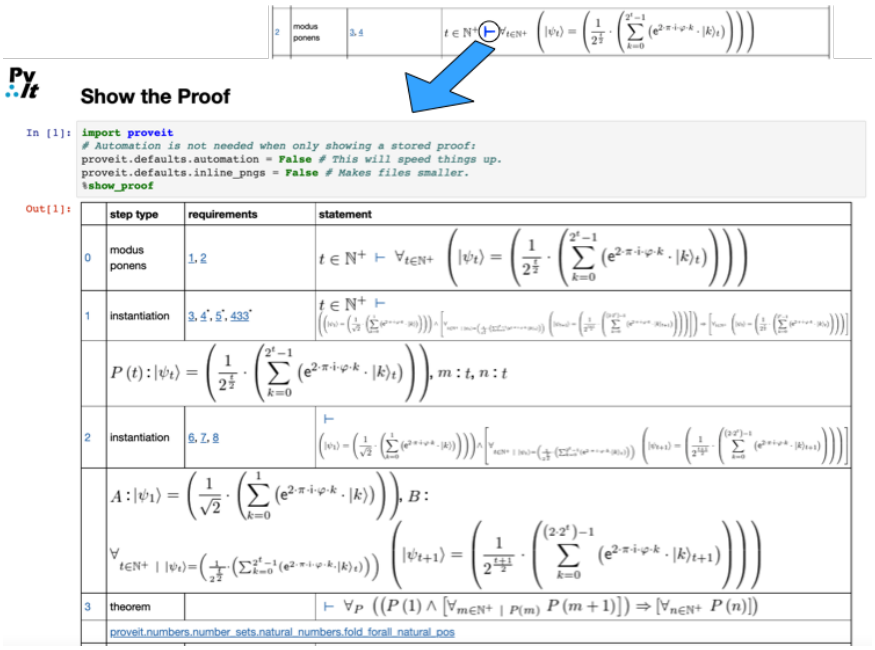
Statements (axioms and theorems) are organized into theory packages in our system. Our universal QPE proofs derive from a number of statements from various packages. Table I lists the number of statements from each theory package (along with some examples of such statements) that, at the time of this 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.[4] As these conjectures are eventually proven, the number of dependencies will decrease, eventually leaving a much smaller number of axiom dependencies.

此外，许多目前作为依赖项出现的公理仅仅是标签定义，因此每一条都充当一种（保守的）"依定义扩展" [38,pp 102–103][43, pp 57–61]。在 **Prove-It** 的未来更新中，我们计划区分保守定义与

[4]此便利功能无需用户采取任何进一步操作，但类似于在 *Coq* 中显式使用 `Accepted` 命令。



(a) The 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. Each numbered step (numbers along the left-hand side) includes an explicit listing (underlined blue numbers in 3rd column) of all previous steps on which the current step depends. Instantiation steps explicitly list the instantiations used.



(b) **Prove-It** 第 2 步形式化证明前几行的截图
前面的图 (5a)，通过点击该行的闸门符号获得的
判断（顶部圈出的内容）。

图 5：在 **Prove-It** 的证明内部获取子证明。

公理。在证明了该标签按其定义唯一存在之后，除非该特定标签出现在该特定定理中，否则在用于证明定理时，该定义将不再构成依赖。数学中常常会偶尔使用方便的记号，例如在

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.

循环依赖在本地...

理论包	# 依赖 公理, 定理	示例
core_expr_types	9, 22	$\lvert() = \lvert(1, 2, .., 0)\rvert$ $\forall_{n\in\mathbb{N}}\left[\forall_{x_1,\ldots,x_n,y_1,\ldots,y_n \mid (x_1,\ldots,x_n)=(y_1,\ldots,y_n)} \left(f\ (x_1,\ldots,x_n)\ = f\ (y_1,\ldots,y_n)\ \right)\right]$
linear_algebra	4, 23	$\forall_K\left[\forall_{V\in_c \mathrm{VecSpaces}(K)}[\forall_{x,y\in V}(x+y\in V)]\right]$ $\forall_K\left[\forall_{V\in_c \mathrm{VecSpaces}(K)}[\forall_{a\in K}[\forall_{x\in V}(ax\in V)]]\right]$
logic.booleans	21, 9	$\forall_{m\in\mathbb{N}^+}\left[\forall_{A_1,\ldots,A_m \mid A_1,\ldots,A_m}\left(A_1\wedge\ldots\wedge A_m\right)\right]$ $\forall_{m\in\mathbb{N}^+}\left[\forall_{A_1,\ldots,A_m \mid A_1,\ldots,A_m}\left(A_1\vee\ldots\vee A_m\right)\right]$
logic.equality	6, 1	$\forall_{x,y,z\mid x=y,y=z}(x=z)$ $\forall_x(x=x)\quad \forall_{x,y}((y=x)=(x=y))$ $\forall_{f,\ x,\ y\mid x=y}\left(f\ (x)\ = f\ (y)\ \right)\quad \forall_{a,\ b,\ c,\ d\mid a=b,\ b=c,\ c=d}(a=d)$
logic.sets	5, 14	$\forall_{A,B}[(A\subseteq B)=(\forall_{x\in A}(x\in B))]$ $\forall_{A,\ B}\left[\ (A\subset B)\ =\ (A\subseteq B\wedge A\neq B)\ \right]$ $\forall_{f,\ a}([\forall_{x,y\in A\mid x\neq y}\ (f\ (x)\neq f\ (y)\ )\ ]\Rightarrow(f\in[A\xrightarrow[\text{onto}]{\text{1-1}}f^{\rightarrow}(A)]))$
numbers.addition	0, 44	$\forall_{A,B\in\mathbb{C}}[A+B\in\mathbb{C}]\quad \forall_{A,B\in\mathbb{C}}[A+B=B+A]\quad \forall_{a\in\mathbb{C}}[a+0=a]$ $\forall_{a,\ x,\ y\in\mathbb{R}\mid x<y}\left[\ (x+a)\ <\ (y+a)\ \right]\quad \forall_{a,b\mid a<b}[a-b<0]$ $\forall_{a,b\mid a\neq b}[a-b\neq 0]\quad \forall_{a,b,c\in\mathbb{C}\mid(a+b)=c}[c-b=a]$
numbers.division	0, 23	$\forall_{a,b\in\mathbb{C}\mid b\neq 0}\left[\frac{a}{b}\in\mathbb{C}\right]\quad \forall_{x\in\mathbb{C}\mid x\neq 0}\left[\frac{x}{x}=1\right]\quad \forall_{x\in\mathbb{C}\mid x\neq 0}\left[\frac{0}{x}=0\right]$ $\forall_{x,y\in\mathbb{X}\mid y\neq 0}\left[\frac{-x}{y}=-\frac{x}{y}\right]\quad \forall_{a,x,y\in\mathbb{R}^+\mid x<y}\left[\frac{a}{x}>\frac{a}{y}\right]$
numbers.exponentiation	0, 41	$\forall_{a\in\mathbb{R}^+,\ b,\ c\in\mathbb{C}}\left[\ (a^b)^{\ c}=a^{bc}\right]\quad \forall_{x\in\mathbb{C}}[x^1=x]\quad \forall_{a,x,y\in\mathbb{R}\mid x=y}[x^a=y^a]$ $\forall_{x\in\mathbb{C}}[\forall_{n\in\mathbb{N}}(x^{2n}=(-x)^{2n})]\quad \forall_{a,b\in\mathbb{C}}[a^b\in\mathbb{C}]$ $\forall_{a,x,y\in\mathbb{R}^+\mid a>1,x<y}[a^x<a^y]$
numbers.multiplication	0, 33	$\forall_{a,b\in\mathbb{C}}[a\cdot b=b\cdot a]\quad \forall_{x\in\mathbb{C}}[1\cdot x=x]\quad \forall_{a,b\in\mathbb{C}}[a\cdot b\in\mathbb{C}]$ $\forall_{a,b\in\mathbb{C}\neq 0}[a\cdot b\in\mathbb{C}^{\neq 0}]\quad \forall_{x,\ y\in\mathbb{C}}\left[\ (-x)\ \cdot y=-\ (x\cdot y)\ \right]$ $\forall_{a,x,y\in\mathbb{R}\mid x<y,a<0}[a\cdot x>a\cdot y]$
numbers.negation	0, 16	$\forall_{a\in\mathbb{C}}[(-a)\in\mathbb{C}]\quad \forall_{a,b\in\mathbb{C}}[-(a+b)=(-a-b)]\quad \forall_{x\in\mathbb{C}}[-(-x)=x]$ $\forall_{x\in\mathbb{C}}[(-1)\cdot x=(-x)]\quad \forall_{x,\ y\in\mathbb{R}\mid x<y}[(-x)>(-y)]\quad -0=0$
numbers.number_sets	2, 83	$\forall_{n\in\mathbb{N}}[(n+1)\in\mathbb{N}]\quad 0\in\mathbb{N}\quad 0\in\mathbb{C}\quad i\in\mathbb{C}\quad \mathbb{R}\subset\mathbb{C}\quad \mathbb{Z}\subset\mathbb{C}$ $\forall_{a,b\in\mathbb{Z}}[\{a,\ldots,b\}\subset\mathbb{Z}]\quad \forall_{a,b\in\mathbb{Z}}[\forall_{n\in\{a,\ldots,b\}}(n\leq b)]\quad \mathbb{N}\subset\mathbb{Z}$
numbers.numerals	2, 36	$1=0+1\quad 2=1+1\quad 0<1\quad 1<2\quad 2\cdot 2=4\quad 3\in\mathbb{N}$ $1\in\mathbb{N}^+\quad 2\in\mathbb{N}^+\quad 3\in\mathbb{N}^+\quad \forall_{a,b,c}(\lvert(a,b,c)\rvert=\lvert(1,2,\ldots,3)\rvert)$
numbers.ordering, numbers.rounding, & numbers.summation	7, 29	$\forall_{x,y,z\mid x<y,y<z}[x<z]\quad \forall_{x\in\mathbb{R}}(\lfloor x\rfloor\leq x)\quad \forall_{x\in\mathbb{R}}(\mathrm{round}(x)\in\mathbb{Z})$ $\forall_{x\in\mathbb{R}}((x-\lfloor x\rfloor)<1)\quad \forall_{x\in\mathbb{R}}(\mathrm{round}(x)=\lfloor x+\frac{1}{2}\rfloor)$ $\forall_f(\forall_{a,\ b,\ c\in\mathbb{Z}}(\sum_{x=a}^{b}f\ (x)\ =\sum_{x=a+c}^{b+c}f\ (x-c)\ ))$
杂项。其他 numbers 子包	0, 29	$\forall_{a,b\in\mathbb{C}}[\lvert a-b\rvert=\lvert b-a\rvert]\quad \forall_{\theta\in\mathbb{R}}(\lvert e^{i\theta}\rvert=1)$ $\left[x\mapsto\frac{1_2}{x}\right]\in\mathtt{MonDecFuncs}(\mathbb{R}^+)\quad \forall_{a,b\in\mathbb{R}^+\mid a\leq b}\left(\left(\int_a^b\frac{1}{\ell^2}d\ell\right)\leq\frac{1}{a}\right)$
physics.quantum	3, 44	$\lvert0\rangle\in\mathbb{C}^2\quad \lvert1\rangle\in\mathbb{C}^2\quad \lvert\lvert\ \lvert0\rangle\ \rvert\rvert=1\quad \lvert\lvert\ \lvert1\rangle\ \rvert\rvert=1\quad \langle0\lvert1\rangle=0$ $\forall_{n\in\mathbb{N}^+}\left(\mathtt{FT}_n^\dagger\in U(2^n)\right)\quad \forall_{n\in\mathbb{N}^+}\left(C^n\in_c\mathtt{HilbertSpaces}\right)$
statistics	1, 7	$\forall_{\Omega\in\mathrm{SampleSpaces}}\left[\forall_{X\in\Omega}\left(\Pr(X)\in[0,1]\right)\right]$ $\forall_{a,\ b\in\mathbb{R}}\left(\lvert e^{ia}-e^{ib}\rvert=2\sin(\frac{\lvert a-b\rvert}{2})\right)$
trigonometry	0, 5	$\forall_{\theta\in\mathbb{R}^+}\left(\sin(\theta)\ <\theta\right)$

表 I：截至本文撰写时，三个主要 QPE 定理 qpe exact (31)、qpe best guarantee (38) 和 qpeprecisionguarantee(53) 在逻辑上所依赖的每个理论包中的公理与（未经证明的）定理总数（以及一些示例）。

level of a formal proof and the broader level of theorem dependencies.

C. QPE证明的长度

As mentioned repeatedly, all of our proofs are viewable on the **Prove-It** website [32]. 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 (shown shaded) and the major supporting theorems, ordered to parallel §II 中的演示。表 III 列出了更多次要定理。图 6 以图形方式展示了表 II 中主要定理和辅助定理之间的相互依赖关系。

The number of steps in a formal proof may be surprising. A lot of details are required in a formal proof. It is also likely 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 §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 interested to

QPE 定理	nb 数量 推导 命令	步骤数 形式化中的 证明
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) $\Omega \in_c$ SampleSpaces	7	29
psi_t_formula (16) $\forall_{t\in\mathbb{N}+}\left(\lvert\psi_t\rangle = \left(\frac{1}{2^{t/2}}\left(\sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}\lvert k\rangle_t\right)\right)\right)$	24	685
alpha_m_evaluation (27) $\forall_{m\in\{0,\ldots,2^t-1\}}\left(\alpha_m = \frac{1}{2^t}\sum_{k=0}^{2^t-1}\left(e^{-\frac{2\pi ikm}{2^t}} e^{2\pi i\varphi k}\right)\right)$	8	404
alpha_ideal_case (30) $\left(2^t\varphi \in \{0,\ldots,2^t-1\}\right) \Rightarrow \left(\alpha_{2^t\varphi} = 1\right)$	2	195
qpe_exact (31)	**15**	**161**
alpha_m_mod_evaluation (32) $\forall_{m\in\mathbb{Z}}\left(\alpha_{m \bmod 2^t} = \frac{1}{2^t}\sum_{k=0}^{2^t-1}\left(e^{-\frac{2\pi ikm}{2^t}} e^{2\pi i\varphi k}\right)\right)$	12	161
alpha_m_mod_as_geometric_sum (33) $\forall_{m\in\mathbb{Z}}\left(\alpha_{m \bmod 2^t} = \frac{1}{2^t}\sum_{k=0}^{2^t-1}\left(e^{2\pi i(\varphi-\frac{m}{2^t})}\right)^k\right)$	8	189
best_guarantee_delta_nonzero (36) $(\delta_{b_r} \neq 0) \Rightarrow \left(\lvert\alpha_{b_r \bmod 2^t}\rvert^2 > \frac{4}{\pi^2}\right)$	23	625
best_guarantee (37) $\lvert\alpha_{\text{round}(2^t\varphi) \bmod 2^t}\rvert^2 > \frac{4}{\pi^2}$	24	259
qpe_best_guarantee (38)	**4**	**19**
alpha_summed (40) $\forall_{\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\mid\ell\neq 0}\left(\alpha_{b_f\oplus\ell} = \frac{1}{2^t}\cdot\frac{1-exp(2\pi i(2^t\delta_{b_f}-\ell))}{1-exp(2\pi i(\delta_{b_f}-\frac{\ell}{2^t}))}\right)$	12	417
fail_sum_prob_conds_equiv_lemma (46)	12	412
fail_sum (45) $\forall_{e\in\{1,\ldots,2^{t-1}-2\}}\left([P_{\text{fail}}](e) = \sum_{\ell=-2^{t-1}+1}^{e} \lvert\alpha_{b_f\oplus\ell}\rvert^2 + \sum_{\ell=e+1}^{2^{t-1}} \lvert\alpha_{b_f\oplus\ell}\rvert^2\right)$	24	453
alpha_sqrd_upper_bound (47) $\forall_{\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\mid\ell\neq 0}\left(\lvert\alpha_{b_f\oplus\ell}\rvert^2 \leq \frac{1}{4(\ell-2^t\delta_{b_f})^2}\right)$	25	706
failure_upper_bound_lemma (48)	16	303
failure_upper_bound (49) $\forall_{e\in\{1,\ldots,2^{t-1}-2\}}\left([P_{\text{fail}}](e) \leq \frac{1}{2e} + \frac{1}{4e^2}\right)$	28	547
success_complements_failure (50) $\forall_{e\in\{1,\ldots,2^{t-1}-2\}}\left([P_{\text{success}}](e) = 1 - [P_{\text{fail}}](e)\right)$	5	70
precision_guarantee_lemma_01 $\left(1 - \frac{1}{2(2^{t-n}-1)} - \frac{1}{4(2^{t-n}-1)^2}\right) > \left(1 - \frac{1}{2(2^{t-n}-1)} - \frac{1}{1-e}\right)$	27	589
precision_guarantee_lemma_02 $\forall_{m\in\{0,\ldots,2^t-1\}\mid\lvert m-b_f\rvert_{2^t}\leq(2^{t-n}-1)}\left(\lvert\frac{m}{2^t} - \varphi\rvert_{\text{mod }1} \leq 2^{-n}\right)$	20	282
precision_guarantee (52)	9	139
qpe_precision_guarantee (53)	**2**	**2**

TABLE II: The 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, and the number of steps (nodes in the formal proof tree) appearing in the corresponding formal proof for the three main QPE theorems (highlighted) and major supporting theorems in the QPE theory package. The ordering of the theorems parallels the overview in §II with theorems depending only on others higher in the list.

注意它与推导命令的数量相比如何。一个给定的推导命令可以生成多个形式化证明步骤，以填补因推导命令序列中跳过显而易见的步骤而产生的空缺。

D. 表达式结构和 LaTeX 格式排版

Prove-It 中的表达式具有内部有向无环图（DAG）结构，这对其解释方式以及允许采用的证明步骤至关重要。LaTeX 排版使表达式更易于人类阅读，但需要注意的是，这种排版-

QPE 定理	# of deriv nb 命令	# 步骤在形式上证明
_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 $\forall_{t\in\mathbb{N}^+}\left(\|\psi_t\rangle \in \mathbb{C}^{2^t} \wedge \| \|\psi_t\rangle \| = 1\right)$	4	329
_Psi_ket_is_normalized_vec $\|\Psi\rangle \in \mathbb{C}^{2^t} \wedge \| \|\Psi\rangle \| = 1$	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 $\left(2^{t-n} - 1\right) \geq 2$	8	249
_e_value_in_e_domain $\left(2^{t-n} - 1\right) \in \{1, \ldots, 2^{t-1} - 2\}$	11	317
_mod_add_closure $\forall_{a,b\in\mathbb{Z}}\left(a \oplus b \in \{0, \ldots, 2^t - 1\}\right)$	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 $\forall_{b\in\{b_f,b_r\}}\left(\delta_b \in (-\frac{1}{2}, \frac{1}{2}]\right)$	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 $\forall_{b\in\{b_f,b_r\}}\left(\delta_b = 0 \vee \delta_b \notin \mathbb{Z}\right)$	15	150
_scaled_delta_b_is_zero_or_non_int $\forall_{b\in\{b_f,b_r\}}\left(2^t\delta_b = 0 \vee 2^t\delta_b \notin \mathbb{Z}\right)$	16	167
_scaled_delta_b_not_eq_nonzeroInt $\forall_{b\in\{b_f,b_r\},\ell\in\mathbb{Z}\|\ell\neq 0}\left(2^t\delta_b \neq \ell\right)$	9	31
_delta_b_not_eq_scaledNonzeroInt $\forall_{b\in\{b_f,b_r\},\ell\in\mathbb{Z}\|\ell\neq 0}\left(\delta_b \neq \frac{\ell}{2^t}\right)$	6	56
_delta_b_floor_diff_in_interval $\forall_{l\in\{-2^{t-1}+1,\ldots,2^{t-1}\}}\left(\delta_{b_f} - \frac{\ell}{2^t} \in [-\frac{1}{2}, \frac{1}{2})\right)$	13	200
_non_int_delta_b_diff $\forall_{b\in\{b_f,b_r\},\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\|\ell\neq 0}\left(\delta_b - \frac{\ell}{2^t} \notin \mathbb{Z}\right)$	14	233
_scaled_abs_delta_b_floor_diff_interval $\forall_{\ell\in\{-2^{t-1}+1,\ldots,2^{t-1}\}\|\ell\neq 0}\left(\pi\|\delta_{b_f} - — \frac{\ell}{2^t}\| \in (0, \frac{\pi}{3}]\right)$	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 $\forall_{b\in\mathbb{Z}}\left(\varphi = \frac{b}{2^t} + \delta_b\right)$	3	43
_modabs_in_full_domain_simp $\forall_{-2^{t-1}+1,\ldots,2^{t-1}}(\|\ell\|_{\text{mod } 2^t} = \|\ell\|)$	6	213

表 III：对于 QPE 理论包中各个较次要的定理，证明笔记本中手动调用的推导命令数量，以及形式化证明中出现的步骤数量。这些次要定理通常涉及简单的代数关系、闭包性质和定义域规范。

matting 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, and their relation with derivation steps, are discussed in Ref. [31]. **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 internal structures. **Prove-It** allows us to present quantum algorithms, and related axioms, definitions, and theorems in quantum circuit form (utilizing Q-Circuit [44]), 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. 7 for an example).

IV. 我们是如何构建证明的?

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

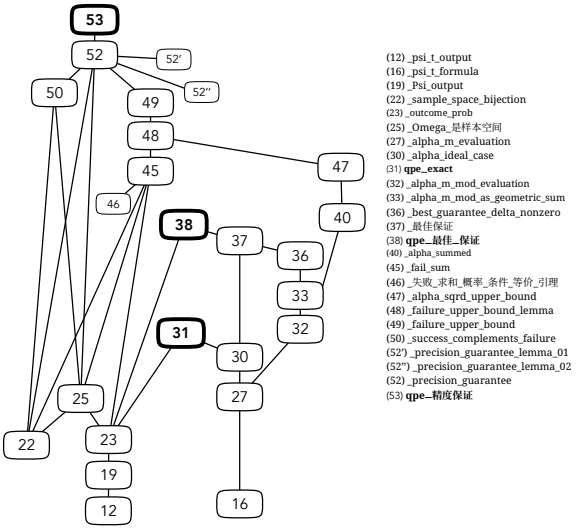


FIG. 6: Graphical depiction of the dependencies among the "main" theorems appearing in Table II. Numbering corresponds to equation numbers in the text (except for the primed numbers appearing in smaller boxes, for lemmas that don't explicitly appear in the text). The three main universal theorems appear in bold. An upward link means that the higher theorem explicitly depends on the lower theorem.

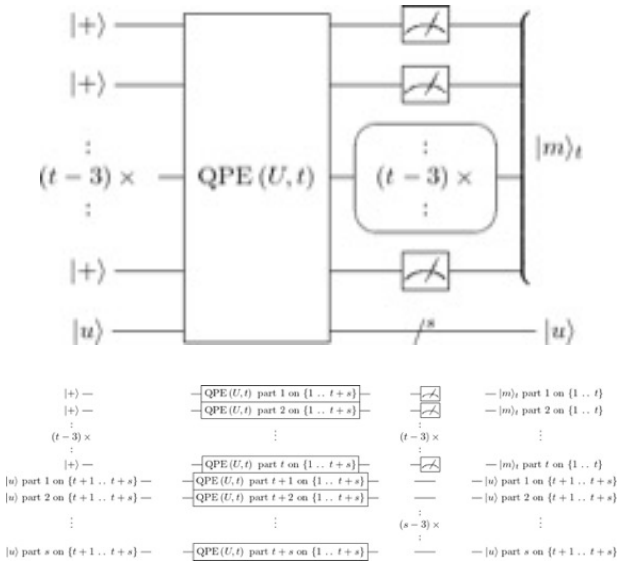


图 7: **Prove-It** 中的一个电路表示示例（上），以及其底层代码表示的风格化版本（下），展示了其组织方式：电路组件按行排列，并嵌套在一系列列中。讨论见 §IIID。

QPE theory package. The system capabilities include:

(1) 公理和定理的手动实例化；

(2) 提供了多种由用户调用的推导命令（策略），这些命令以间接方式执行实例化，以实现更高层次的目的（例如，分配、因式分解、推导新的界限等）；

(3)目标驱动的推导 可被调用来直-

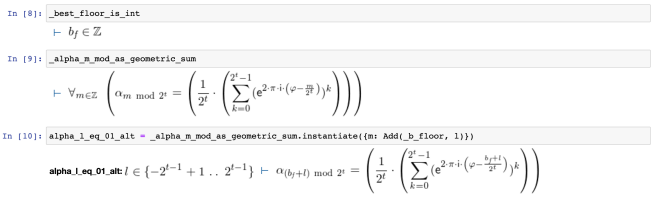


图 8：手动实例化示例，摘自alpha summed (16)，用 $m = (b_f + \ell)$实例化alpha m mod as geometric sum (33)。

直接或间接地；

(4)使用规范形式来识别等价表达式并指导自动证明步骤；(5) 每当建立判断或作出假设时，都会执行完全自动的附带推导；(6)以及广泛且可由用户配置的表达式自动化简 。

正如示例所示，大多数证明步骤都涉及这些能力的协同运作。

A. 手动实例化

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.*, $x_1 \cdot y_1 + \ldots + x_n \cdot y_n$), automatically attempt to prove required conditions of the quantifiers as needed (see § IV C), and take advantage of automatic simplification of the instantiated formula (see § IV F), and otherwise providing an informative error message (specifying, *e.g.*, that some instantiation condition could not be proven).

An example of manual (*i.e.*, explicitly user-invoked) instantiation is shown in Fig. 8, 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]`.

图 9展示了另外两个手动实例化的示例，这些示例摘录自关于psit 公式 (16)的证明笔记。单元格 `[4]` 和 `[5]` 展示了关于正自然数的一般归纳定理，以及针对 psi t 公式定理的具体实例化。

（变量 m 和 n 分别实例化为 t，函数 P 则实例化为求和公式）。单元格[8] 和 [9] 分别显示 psi t def (13) 及其针对 $t = 1$的实例化。

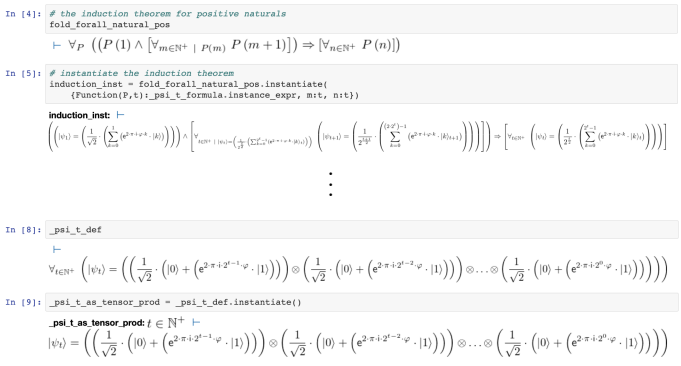
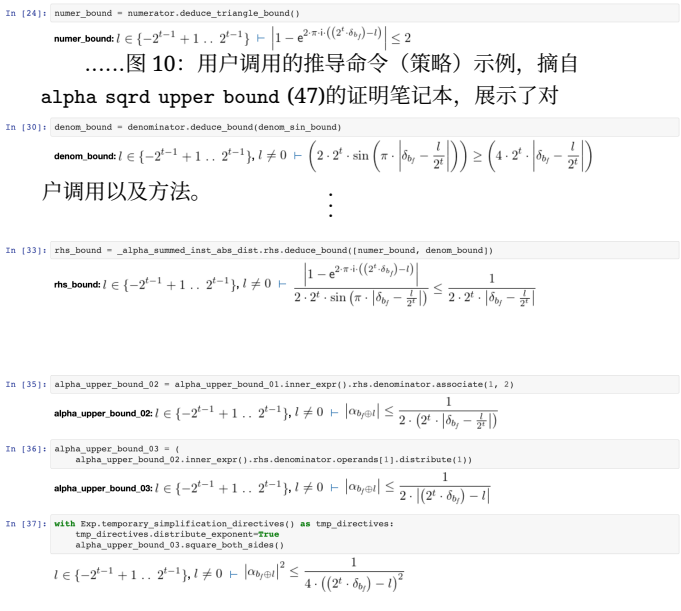


FIG. 9: Examples of manual instantiations, excerpted from the proof notebook for the `_psi_t_formula` (16), showing the manual instantiation of the induction theorem for positive naturals (`cells 4` and `5`) and of the `_psi_t_def` (13) local QPE definition (`cells 8` and `9`).

B. 用户调用的推导（命令/战术）

用户调用的推导通常会使用 Expression 类方法，其中包含用于确定要调用哪些公理/定理的指令，并且 ap-



.....图 10：用户调用的推导命令（策略）示例，摘自 `alpha sqrd upper bound` (47)的证明笔记本，展示了对户调用以及方法。

propriate instantiations of their quantified variables.[5] This frees the user from having to find the right axiom(s)/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 post-processing of resulting judgments.

Some examples of such user-invoked derivations are shown in Fig. 10, excerpted from the proof notebook for `_alpha_sqrd_upper_bound` (47). The `NumberOperation` class `deduce_bound()` method, employed in cells `[30]` and `[33]`, is a particularly powerful method. It propagates the effects of any number of bounds on sub-expressions. Under the hood, the `deduce_bound()` step in cell `[33]` also employs goal-driven derivations (see § IV C) to satisfy requirements (*e.g.*, establishing the signs of the numerator and denominator) as well as automatic simplification (see § IV F) in canceling out a factor of 2.

C. 目标驱动的推导

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.

在单元格 [10]的实例化步骤中，已经遇到过间接（自动）目标驱动推导。

[5] 底层代码的面向对象组织方式也让查找所需的方法变得容易。

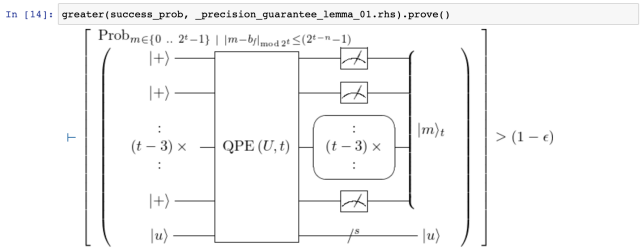


图 11：目标驱动推导示例，摘自 `precision guarantee`(52) 的证明笔记本，展示了如何手动对感兴趣的公式调用 `.prove()` 。

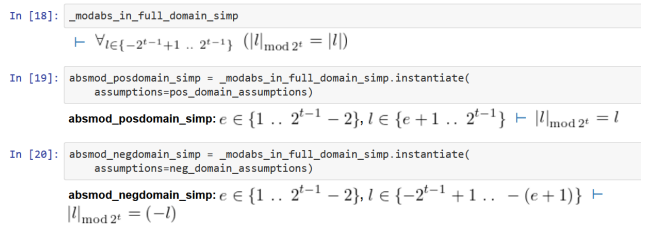


图 12： `fail sum` (45)) 证明笔记本摘录中的偶然推导示例。

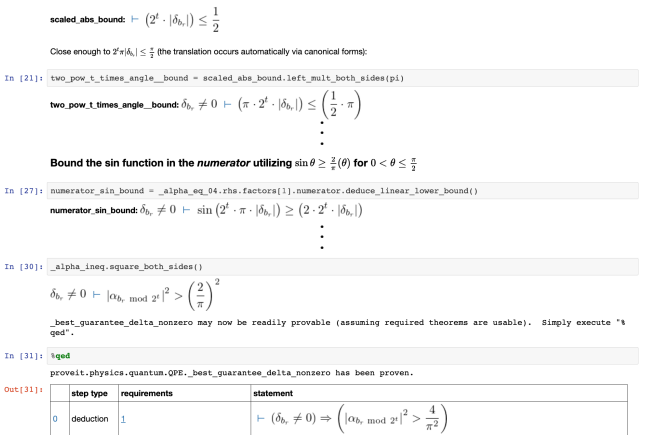


图 13：允许自动推导步骤的等价规范形式示例，摘录自 `best guarantee delta nonzero` (36) 的证明笔记本。

in Fig. 8, with the system proving "under the hood" 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. 11, 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 ($<$, $\leq$, 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 $c \leq d$). The same bidirectional search algorithm can be utilized in goal-driven derivations of other transitive relations (e.g., subset relations: $\subset$, $\subseteq$, and $\cong$).

D. 附带推导

许多简单的推导会在无需用户指示的情况下自动执行，以主动推导出可能对后续推导有用的事实。证明判断或作出假设时，通常也会进行附带推导。这些附带推导由各个 Expression 类负责。

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 $A \vdash x \in \{2 \,..\, 10\}$ will incidentally derive (in the background, so to speak) $A \vdash x \geq 2$ and $A \vdash x \leq 10$ (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.

图 12 的单元格 `[19]`, `[20]` 中展示了一对相关的附带推导示例（摘自等式（45）求和失败的证明笔记本）。在每个示例中—ple, assumptions generate incidental derivations that enable an instantiation and its simplification. We assume $e \in \{1 \,..\, 2^{t-1} - 2\}$ in both cases which incidentally derives $e \geq 1$ and $e \leq 2^{t-1} - 2$. In cell `[19]`, we additionally assume $l \in \{e+1 \,..\, 2^{t-1}\}$ which incidentally derives $l \geq e + 1$ and $l \leq 2^{t-1}$ from which it can automatically derive $l \in \{-2^{t-1} + 1 \,..\, 2^{t-1}\}$ 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 §IV F). In cell `[20]`, we additionally assume $l \in \{-2^{t-1} + 1 \,..\, -(e+1)\}$ from which we can also derive $l \in \{-2^{t-1} + 1 \,..\, 2^{t-1}\}$ 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 bi-directional fashion and can be very powerful.

E. 将具有相同规范形式的表达式等同起来

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 presenta-

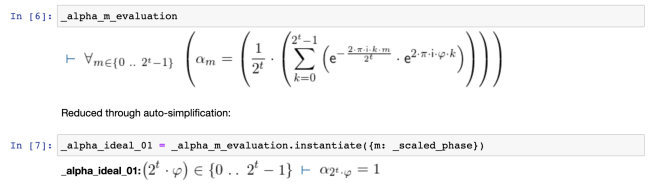


图 14：自动化简示例，摘自 `alpha ideal case` (30) 的证明笔记。

tion of the ranges). Another example is the total-ordering style of a conjunction of transitive relations; for example, $(a < b) \wedge (b < c)$ 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. $(2\cdot 3+x)$, 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. 13, excerpted 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 $\theta = 2^t\pi|\delta_{b_r}| \leq \frac{\pi}{2}$, which the system recognizes through canonical forms as equivalent to the inequality derived in cell `[21]`. Later, the interactive steps have established a formula in cell `[30]` involving the expression $\left(\frac{2}{\pi}\right)^2$, while the theorem to be proven uses the expression $\frac{4}{\pi^2}$. While $\left(\frac{2}{\pi}\right)^2$ 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. 自动简化

Why not automatically simplify everything to its canonical form? This is not always desirable and sometimes counter-productive. 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 可配置——也就是说，用户可以手动禁用自动简化，或显式更改每种表达式类型的默认简化指令。

Automatic simplifications are performed as a post-processing 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 sub-expressions 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. 14, 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 auto-simplification. 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 auto-simplification saves many steps throughout our QPE derivations.

V. 检查 QPE 软件包中的一个特定证明

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 [32]. There is too much material there to include in this article. But 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 Fig. 15. While many steps are omitted for brevity, these chosen steps show the basic flow of this inductive proof and illustrate how derivation commands (discussed in §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. 16. 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**

正在补充证明构造中省略的显而易见的步骤。

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 Appendix A, 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 Appendix A with the 24 derivation commands used to prove `_psi_t_formula`. This ratio is analogous to what de Bruijn called a "loss factor" [45] and Wiedijk has called "the de Bruijn factor" [46]. 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 §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. 17.

VI. 与其他方法的比较

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 Liu, *et al.* [27], Hietala, *et al.* [36], Chareton, *et al.* [23], and Bauer-Marquart, *et al.* [47]. Our work is unique in providing a verification for Nielsen & Chuang's [30] 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.* [25] implemented quantum Hoare logic (QHL) [28] 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 [36], this is "then only a partial verification effort." Also, they do not prove any precision guarantees on the phase estimation in QPE.

Using SQIR ("small quantum intermediate representation"), which they describe as "a simple quantum language deeply embedded in the Coq proof assistant," Hietala, *et al.* [36] 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 (QFT), quantum phase estimation (QPE), 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. [36] and can be found in their SQIR repository. There are immediate differences in 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. And 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 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 the top panel of Fig. 18; the specification in **Prove-It**, including the automatically-generated graphical circuit representation, appears in the bottom panel. **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** was 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).[6]

[6] Coq 命令 were 已枚举 for Hietala, *et*

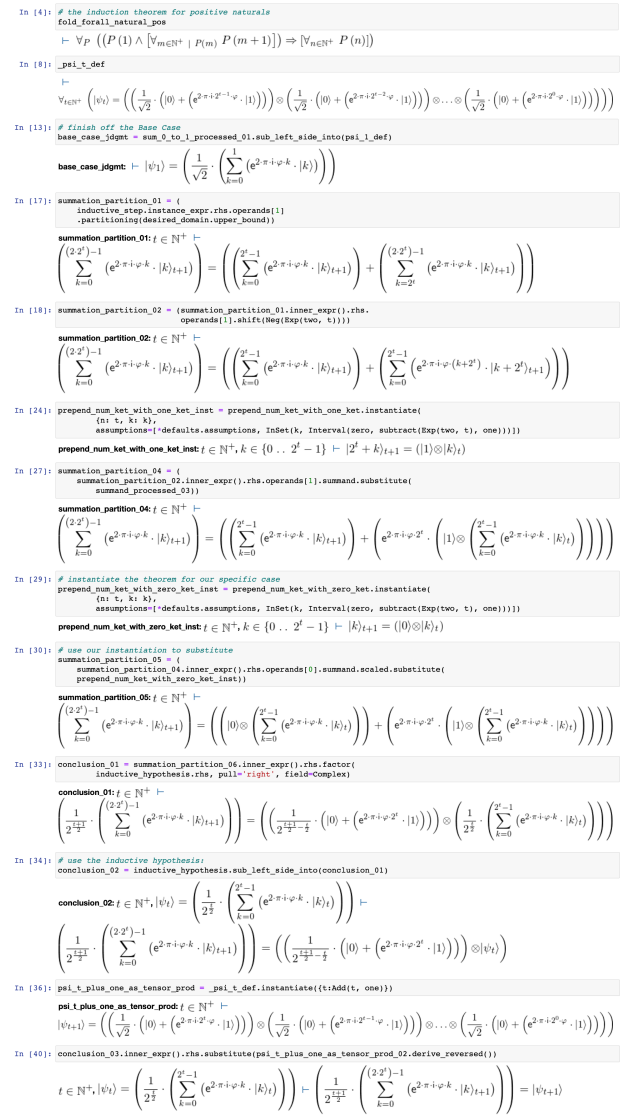


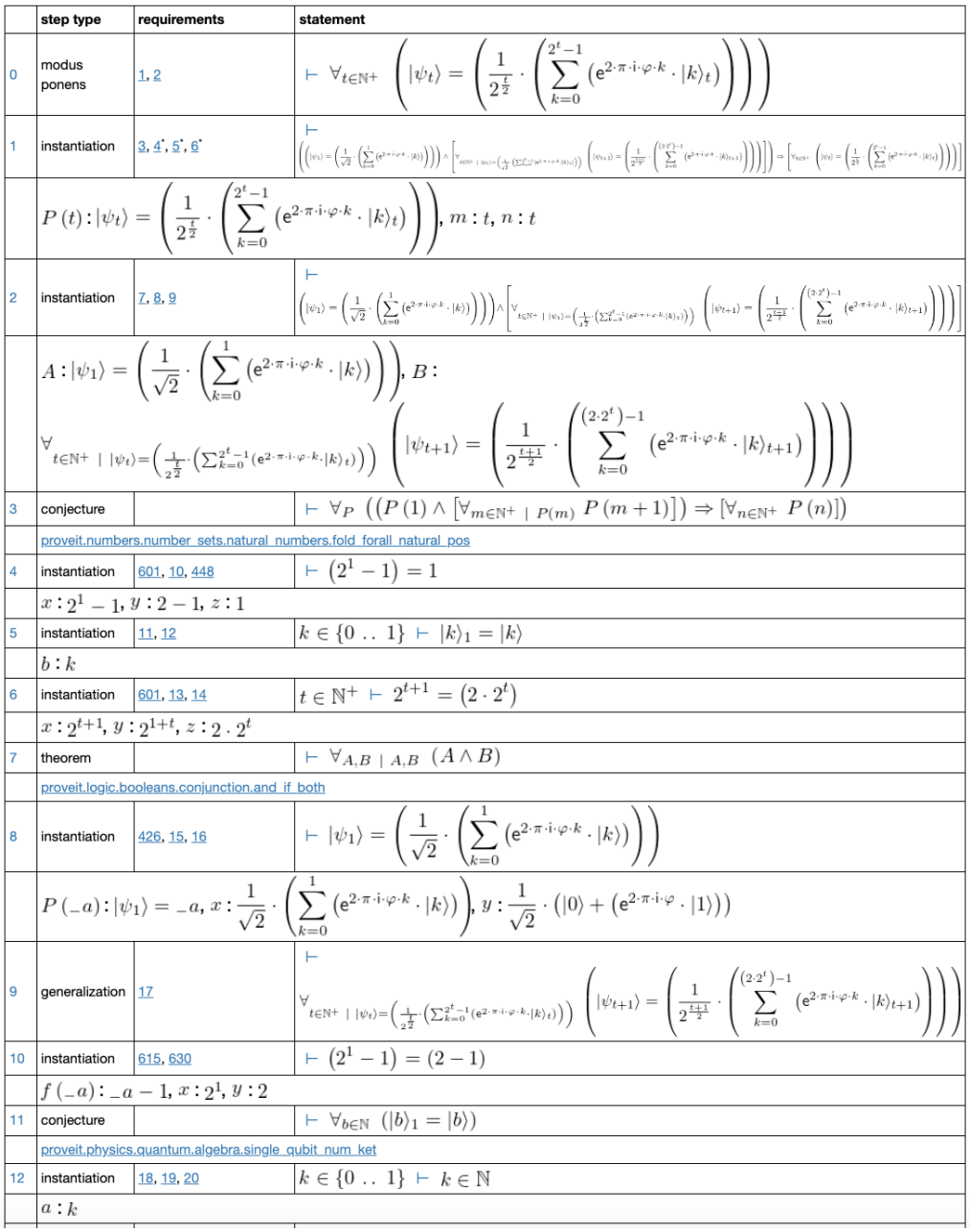
FIG. 15: A subset (13 of 23) of the interactive steps that construct the proof of `_psi_t_formula` (16) chosen to show the basic flow of this inductive proof and demonstrate some useful derivation commands in action such as **`partitioning`**, **`shift`**, **`substitute`**, and **`factor`**. Appendix A compares this to a structured proof format intended to communicate a proof among experts.

This comparison is presented in Table IV. We cannot make a comparison for the most challenging proof since this particular form was not proven in Ref. [36]. 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 条 **Prove-It** 推导命令，而 qpe 精确性定理需要 137 条 Coq 命令；qpe 最佳保证定理则分别需要 213 条和 313 条命令。

In his Ph.D. thesis, Amy used the Feynman path integral model of quantum circuits for their optimization, synthesis and functional verification [48]. 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 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

al.'s `QPE_simplify`, `QPE_semantics_simplified`, and `QPE_full_semantics` lemmas [36], 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 semi-colon. The Prove-It derivation commands tally the numbers in Table II and II according to the dependencies of the three high-level statements respectively as shown in Fig. 6.



	step type	requirements	statement
0	modus ponens	1, 2	$\vdash \forall_{t\in\mathbb{N}^+} \left(\|\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot \|k\rangle_t\right)\right)\right)\right)$
1	instantiation	3, 4*, 5*, 6*	$\vdash$
	$P(t): \|\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}} \cdot \left(\sum_{k=0}^{2^t-1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot \|k\rangle_t\right)\right)\right), m: t,\ n: t$		
2	instantiation	7, 8, 9	$\vdash$
	$A: \|\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot \|k\rangle\right)\right)\right), B:$		
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	$k \in \{0\ ..\ 1\} \vdash \|k\rangle_1 = \|k\rangle$
	$b: k$		
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		$\vdash \forall_{A,B \mid A,B} \left(A \wedge B\right)$
	proveit.logic.booleans.conjunction.and_if_both		
8	instantiation	426, 15, 16	$\vdash \|\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot \|k\rangle\right)\right)\right)$
	$P(_a): \|\psi_1\rangle = _a,\ x: \frac{1}{\sqrt{2}} \cdot \left(\sum_{k=0}^{1} \left(e^{2\cdot\pi\cdot i\cdot\varphi\cdot k} \cdot \|k\rangle\right)\right),\ y: \frac{1}{\sqrt{2}} \cdot \left(\|0\rangle + \left(e^{2\cdot\pi\cdot i\cdot\varphi} \cdot \|1\rangle\right)\right)$		
9	generalization	17	$\vdash$
10	instantiation	615, 630	$\vdash \left(2^1 - 1\right) = (2 - 1)$
	$f(_a): _a - 1,\ x: 2^1,\ y: 2$		
11	conjecture		$\vdash \forall_{b\in\mathbb{N}} \left(\|b\rangle_1 = \|b\rangle\right)$
	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. 16: The 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).

伪布尔相位函数具有关于变量的多线性多项式的唯一表示。一套针对多项式表示的重写规则演算被用于计算与量子电路相关联的范式多项式，该多项式可以在多项式时间内计算；然而，这些范式并不

唯一的。可逆函-数的量子算法/电路，尤其是使用Clifford+T 门实现的Toffoli门和加法器，均得到了验证；包括量子傅里叶变换和量子隐藏移位算法在内的更多非- 平凡算法也得到了验证。因此，验证归结为检查电路是否等价-

定理	**Prove-It** 推导命令 总复用量		附加的	*Coq* 命令 总计	复用	额外
qpe 精确	87	—	—	137	—	—
QPE 最佳保证	213	72	141	313	73	240
qpe 精度保证 437		145	292	—	—	—

TABLE IV: Number of **Prove-It** derivation command steps vs. Coq commands [36] for the main QPE theorems. The "reused" and "additional" steps add to the left-column total, providing a measure of how many derivation steps in later theorems are reused from earlier theorems.

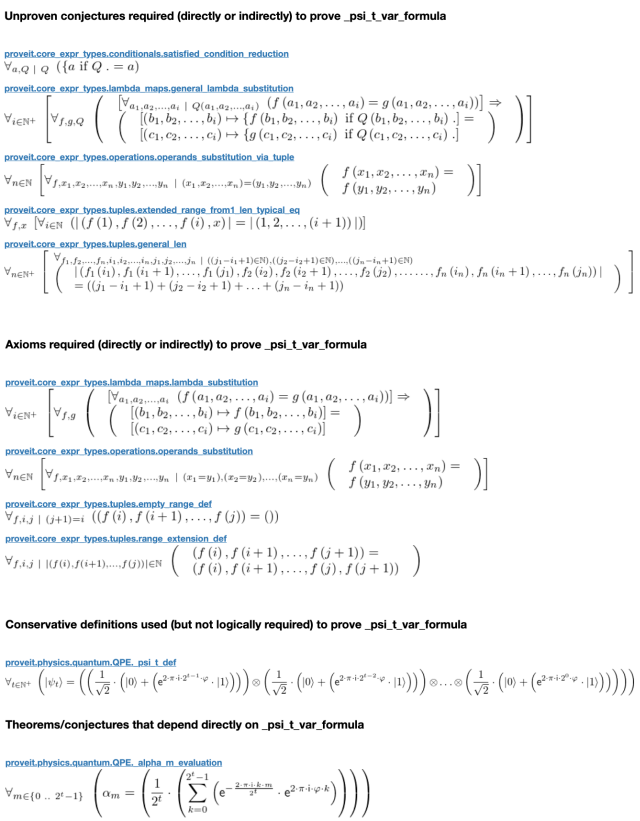


FIG. 17: Excerpt from the dependencies page `_psi_t_formula` (16), showing some of the unproven theorems, axioms, and conservative definitions on which the theorem depends, and `_alpha_m_evaluation` (27) which depends directly on the `_psi_t_formula` theorem itself. Such dependencies are all explicitly tracked in **Prove-It** and readily accessible for review by the user.

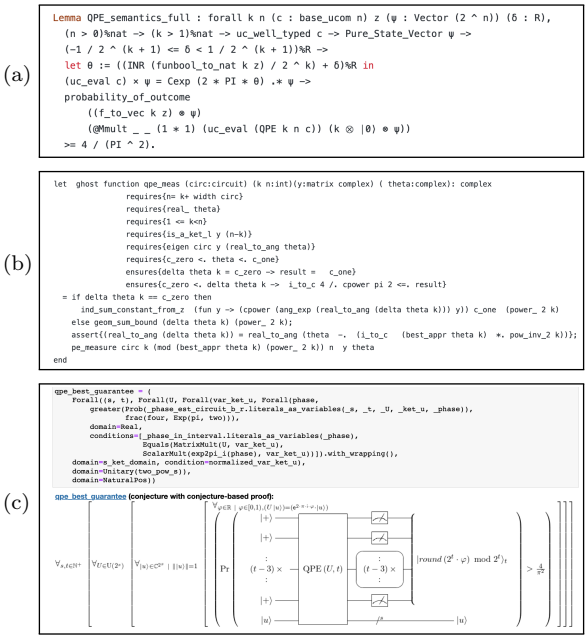


图 18： `qpe best guarantee`(38) 在 (*a*) SQIR [36] *vs.* (*b*) QBricks [15] *vs.* (*c*) **Prove-It** 中的规范。**Prove-It** 中的规范会自动生成相应的电路表示，有助于提高呈现的清晰度与人类可读性。

等价性是通过计算电路的规范形来完成的。该方法在用于量子电路的量子编译器后端基础设施FEYNMAN 中实现，类似于编程语言中的LLVM [49, 50]。由于 Amy 的工作未考虑测量门，因此省略了相位估计的精度保证。

使用他们在 Why3 演绎验证环境 [51], 中构建的 Qbricks 量子电路程序环境 [24]， Chareton等人 [23] 对 Grover 量子搜索算法的实现进行编码和验证，

the Quantum Fourier Transform (QFT), Quantum Phase Estimation (QPE), and Shor's algorithm. They used Amy's Feynman path sum model of quantum circuits as building blocks/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 product and unity circle, etc. They also generalized Amy's work to implement parameterized path sums for modeling quantum

具有任意数量量子比特的电路。

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 Hietala, *et al.*'s [36] use of SQIR, 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 the middle panel of Fig. 18. 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 [30] which we describe in §VII. More generally, we would argue that **Prove-It** offers advantages in flexibility and relative ease of use over the Qbricks and SQIR/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. 讨论与结论

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 a new 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 (QPE). 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/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 check-able. 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 [30]. 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 check-able 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/diagrammatic notation for human-readable theorems and proofs. We also noted the we compare favorably against a Coq implementation for proving QPE in the number of derivation/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 [30] and eventually led to a better bound on the precision guarantee. For example, in their section §5.2.1 on "Performance & Requirements," Nielsen & Chuang cite the constraint $0 \leq \delta \leq 2^{-t}$ whereas $0 \leq \delta < 2^{-t}$ (with the strict inequality on the right) is actually needed, and they use $-\pi \leq 2\pi(\delta - \ell/2^t) \leq \pi$ where $-\pi \leq 2\pi(\delta - \ell/2^t) < \pi$ (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 & Chuang's Eq. 5.29, and the equal sign in their Eq 5.34 should be a less-than. 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 $p\big(|m - b| > e\big) < \frac{1}{2e} + \frac{1}{4e^2}$ instead of $\frac{1}{2(1-e)}$ and allowing $e \geq 1$ instead of just $e \geq 2$.

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. This current work demonstrates what is possible with this approach. But, 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 thoughputs.

VIII. 致谢

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 [52] 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 (QCAT) and Quantum Systems Accelerator (QSA) programs, and the Laboratory Directed Research and Development program at Sandia National Laboratories. 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 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 United States Government

附录 A：结构化证明示例

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 [39, 40]. 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, Robert Carr, 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 §V).

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

假设

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.*, $\psi_t = p_{t-1}\otimes p_{t-2}\otimes\ldots\otimes p_0$.)
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. $\psi_{t-1} = \frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}|k\rangle_t$ iff $\psi'_{t-1} = \sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}|k\rangle_t$.
11. 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. $\forall_{t\in\mathbb{N}^+}\Big[\psi_t = \frac{1}{2^{t/2}}\Big((|0\rangle + e^{2\pi i 2^{t-1}\varphi}|1\rangle)\otimes (|0\rangle + e^{2\pi i 2^{t-2}\varphi}|1\rangle)\otimes\cdots\otimes(|0\rangle + e^{2\pi i 2^{0}\varphi}|1\rangle)\Big)\Big]$.
13. $\forall_{t\in\mathbb{N}^+}\Big[f(t) = \frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}|k\rangle_t\Big]$.
14. $\forall_{t\in\mathbb{N}^+}\ [P(t)\Leftrightarrow\psi_t = f(t)]$
15. $\psi_1 = \frac{1}{2^{1/2}}\left(|0\rangle + e^{2\pi i\varphi}|1\rangle\right)$
16. $f(1) = \frac{1}{2^{1/2}}\left(|0\rangle_1 + e^{2\pi i\varphi}|1\rangle_1\right)$
17. $\forall_{a,b,c\in\mathbb{Z}}\forall_\alpha\left(\sum_{k=a}^{c}\alpha_k = \sum_{k=a}^{b}\alpha_k + \sum_{k=b+1}^{c}\alpha_k\right)$
18. $\forall_{k\in\{2^{t'},2^{t'}+1,\ldots,2^{t^*}-1\}}\left[e^{2\pi i\varphi k}|k\rangle_{t^*} = e^{2\pi i\varphi(k-2^{t'}+2^{t'})}\left|k-2^{t'}+2^{t'}\right\rangle_{t^*}\right]$

结构化证明（对 t 进行归纳）：

基础情形。

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

归纳步骤。

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

5. $\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} =$
$\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} + \sum_{k=2^{t'}}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$
and $\sum_{k=2^{t'}}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} = \sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k+2^{t'})}\left|k+2^{t'}\right\rangle_{t^*}$.
Thus $\sum_{k=0}^{2^{t^*}-1} e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$
$= \sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} + \sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k+2^{t'})}\left|k+2^{t'}\right\rangle_{t^*}$.
6. $\forall_a \forall_b \forall_J\ \sum_{j\in J} a\otimes b_j = a\otimes\sum_{j\in J} b_j$
and $\forall_{t\in\mathbb{N}^+}\forall_{k\in]2^t[}\ \left|1\right\rangle\otimes\left|k\right\rangle_t = \left|2^t+k\right\rangle_{t+1}$.
Thus $\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(2^{t'}+k)}\left|2^{t'}+k\right\rangle_{t^*}$
$= e^{2\pi i\varphi(2^{t'})}\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k)}\ \left|1\right\rangle\otimes\left|k\right\rangle_{t'}$
$= \left|1\right\rangle\otimes\left(e^{2\pi i\varphi(2^{t'})}\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k)}\left|k\right\rangle_{t'}\right)$
7. $\forall_a \forall_b \forall_J\ \sum_{j\in J} a\otimes b_j = a\otimes\sum_{j\in J} b_j$
and $\forall_{t\in\mathbb{N}^+}\forall_{k\in[2^{t-1}]}\ \left|0\right\rangle\otimes\left|k\right\rangle_t = \left|k\right\rangle_{t+1}$.
Thus $\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$
$= \sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|0\right\rangle\otimes\left|k\right\rangle_{t'} = \left|0\right\rangle\otimes\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t'}$.
8. $\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$
$= \sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} + \sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k+2^{t'})}\left|k+2^{t'}\right\rangle_{t^*}$
and $\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(2^{t'}+k)}\left|2^{t'}+k\right\rangle_{t^*}$
$= \left|1\right\rangle\otimes e^{2\pi i\varphi(2^{t'})}\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k)}\left|k\right\rangle_{t'}$
and $\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} = \left|0\right\rangle\otimes\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t'}$.
Thus $\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$
$= \left|0\right\rangle\otimes\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t'} + \left|1\right\rangle$
$\otimes e^{2\pi i\varphi(2^{t'})}\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k)}\left|k\right\rangle_{t'}$
$= \left(\left|0\right\rangle + \left|1\right\rangle e^{2\pi i\varphi(2^{t'})}\right)\otimes\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k)}\left|k\right\rangle_{t'}$.
9. $\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$
$= \left(\left|0\right\rangle + \left|1\right\rangle e^{2\pi i\varphi(2^{t'})}\right)\otimes\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi(k)}\left|k\right\rangle_{t'}$
and $\forall_{k\in]t[}\ p'_k = (\left|0\right\rangle + e^{2\pi i 2^k\varphi}\left|1\right\rangle)$
and $\psi'_{t'} = \sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t'}$.
Thus $\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} = p'_{t'}\otimes\psi'_{t'}$
10. $\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} = p'_{t'}\otimes\psi'_{t'}$ and $\psi'_1 = p'_0$
and $\forall_{k\in[t-1]}\ \psi_{k+1} = p'_k\otimes\psi'_k$.
Thus $\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} = \psi'_{t^*}$
11. $\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*} = \psi'_{t^*}$ and $[\forall_{a,b,\alpha}$ if $a=b$ then $\alpha a = \alpha b]$
and $a = \sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$ and $b = \psi'_{t^*}$ and $\alpha = \frac{1}{2^{t^*/2}}$.
Thus $\frac{1}{2^{t^*/2}}\psi'_{t^*} = \frac{1}{2^{t^*/2}}\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$
12. $\frac{1}{2^{t^*/2}}\psi'_{t^*} = \frac{1}{2^{t^*/2}}\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$ and $\frac{1}{2^{t^*/2}}\psi'_{t^*} = \psi_{t^*}$.
Thus $\psi_{t^*} = \frac{1}{2^{t^*/2}}\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$.
13. Assuming $\psi_{t'} = \frac{1}{2^{t'/2}}\sum_{k=0}^{2^{t'}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t'}$
and t' was arbitrary,
we obtain $\psi_{t^*} = \frac{1}{2^{t^*/2}}\sum_{k=0}^{2^{t^*}-1}\ e^{2\pi i\varphi k}\left|k\right\rangle_{t^*}$.
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, IBM Quantum (n.d.), `https://quantum-computing.ibm.com/`. Last accessed on 3/04/2023.
[2] Rigetti Computing, IBM Q (2023), `https://www.rigetti.com/`. Last accessed on 3/04/2023.
[3] Oxford Quantum Circuits, (2023), `https://oxfordquantumcircuits.com/technology/`. Last accessed on 3/04/2023.
[4] Quantum Inspire, Starmon-5 (2023), `https://www.quantum-inspire.com/backends/starmon-5/`. Last accessed on 3/04/2023.
[5] IonQ, Trapped Ion Quantum Computing (2023), `https://ionq.com/`. Last accessed on 3/04/2023.
[6] Sandia National Laboratories, QSCOUT (2023), `https://www.sandia.gov/quantum/quantum-information-sciences/projects/qscout/`. Last accessed on 3/04/2023.
[7] QuEra, Neutral Atom Platform (2022), `https://www.quera.com/neutral-atom-platform`. Last accessed on 3/04/2023.
[8] Xanadu, Photonics (2023), `https://xanadu.ai/photonics/`. Last accessed on 3/04/2023.
[9] Quandela, Quandela Cloud (2022), `https://cloud.quandela.com/`. Last accessed on 3/04/2023.
[10] Quantum Inspire, Spin2 (2023), `https://www.quantum-inspire.com/backends/spin-2/`. Last accessed on 3/04/2023.
[11] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994)

步骤	判断	原因	笔记本 cell
0	$\vdash \forall_{t\in\mathbb{N}^+}\ \left(\lvert\psi_t\rangle = \left(\frac{1}{2^{\frac{t}{2}}}\cdot\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\cdot\lvert k\rangle_t\right)\right)\right)\right)$	3, 8, 17	
3	$\vdash \forall_P\ \left(\left(P(1)\wedge\left[\forall_{m\in\mathbb{N}^+\ \mid\ P(m)}\ P(m+1)\right]\right)\Rightarrow\left[\forall_{n\in\mathbb{N}^+}\ P(n)\right]\right)$	归纳定理	4
8	$\vdash \lvert\psi_1\rangle = \frac{1}{\sqrt{2}}\sum_{k=0}^{1}\left(e^{2\pi i\varphi k}\lvert k\rangle_1\right)$	101	13
17	$t\in\mathbb{N}^+, \lvert\psi_t\rangle = \left(\frac{1}{2^{t/2}}\left(\sum_{k=0}^{2^t-1}\left(\text{和}^{2\pi i\varphi k}\lvert k\rangle_t\right)\right)\right)\vdash \lvert\psi_{t+1}\rangle = \left(\frac{1}{2^{\frac{t+1}{2}}}\left(\sum^{(2\cdot 2^t)}_{k=0}{}^{-1}\left(\mathsf{e}^{2\pi i\varphi k}\lvert k\rangle_{t+1}\right)\right)\right)$	47, 84, 101	40
47	$t\in\mathbb{N}^+, \lvert\psi_t\rangle = \left(\frac{1}{2^{t/2}}\left(\sum_{k=0}^{2^t-1}\left(\text{和}^{2\pi i\varphi k}\lvert k\rangle_t\right)\right)\right)\vdash \left(\frac{1}{2^{\frac{t+1}{2}}}\left(\sum^{(2\cdot 2^t)}_{k=0}{}^{-1}\left(\mathsf{e}^{2\pi i\varphi k}\lvert k\rangle_{t+1}\right)\right)\right) = \left(\left(\frac{1}{\sqrt{2}}\left(\lvert 0\rangle + \left(\text{和}^{2\pi i\varphi 2^t}\lvert 1\rangle\right)\right)\right)\otimes\lvert\psi_t\rangle\right)$	60	34
60	$t\in\mathbb{N}^+\vdash \left(\frac{1}{2^{\frac{t+1}{2}}}\cdot\left(\sum^{(2\cdot 2^t)}_{k=0}{}^{-1}\left(\mathsf{e}^{2\pi i\varphi k}\lvert k\rangle_{t+1}\right)\right)\right) = \left(\left(\frac{1}{\sqrt{2}}\left(\lvert 0\rangle + \left(\text{和}^{2\pi i\varphi 2^t}\lvert 1\rangle\right)\right)\right)\otimes\left(\frac{1}{2^{t/2}}\cdot\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\pi\mathsf{i}\varphi k}\lvert k\rangle_t\right)\right)\right)\right)$	92	33
84	$t\in\mathbb{N}^+\vdash \lvert\psi_{t+1}\rangle = \left(\begin{array}{l}\left(\frac{1}{\sqrt{2}}\cdot\left(\lvert 0\rangle + \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot 2^t\cdot\varphi}\cdot\lvert 1\rangle\right)\right)\right)\otimes \\ \left(\frac{1}{\sqrt{2}}\cdot\left(\lvert 0\rangle + \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot 2^{t-1}\cdot\varphi}\cdot\lvert 1\rangle\right)\right)\right)\otimes \\ \ldots\otimes\left(\frac{1}{\sqrt{2}}\cdot\left(\lvert 0\rangle + \left(\text{和}^{2\cdot\pi\cdot i\cdot 2^0\cdot\varphi}\cdot\lvert 1\rangle\right)\right)\right)\end{array}\right) = \left(\sum^{(2\cdot 2^t)}_{k=0}{}^{-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\cdot\lvert k\rangle_{t+1}\right)\right) =$	101	36
92	$t\in\mathbb{N}^+\vdash \left(\begin{array}{c}\left(\lvert 0\rangle\otimes\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\cdot\lvert k\rangle_t\right)\right)\right)+ \\ \left(\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot 2^t}\cdot\lvert 1\rangle\right)\otimes\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\cdot\lvert k\rangle_t\right)\right)\right)\end{array}\right)$	142, 178, 181, 218, 350	30
101	$\vdash \forall_{t\in\mathbb{N}^+}\ \left(\lvert\psi_t\rangle = \left(\begin{array}{l}\left(\frac{1}{\sqrt{2}}\cdot\left(\lvert 0\rangle + \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot 2^{t-1}\cdot\varphi}\cdot\lvert 1\rangle\right)\right)\right)\otimes \\ \left(\frac{1}{\sqrt{2}}\cdot\left(\lvert 0\rangle + \left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot 2^{t-2}\cdot\varphi}\cdot\lvert 1\rangle\right)\right)\right)\otimes \\ \ldots\otimes\left(\frac{1}{\sqrt{2}}\cdot\left(\lvert 0\rangle + \left(\mathsf{e}^{2\cdot\pi\cdot i\cdot 2^0\cdot\varphi}\cdot\lvert 1\rangle\right)\right)\right)\end{array}\right)\right) = \left(\sum^{(2\cdot 2^t)}_{k=0}{}^{-1}\left(\mathsf{e}^{2\cdot\pi\cdot\mathsf{i}\cdot\varphi\cdot k}\cdot\lvert k\rangle_{t+1}\right)\right) =$	定义	8
142	$t\in\mathbb{N}^+\vdash \left(\begin{array}{c}\left(\sum_{k=0}^{2^t-1}\left(\text{和}^{2\cdot\pi\cdot i\cdot\varphi\cdot k}\cdot\lvert k\rangle_{t+1}\right)\right)+ \\ \left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot i\cdot\varphi\cdot(k+2^t)}\cdot\lvert k+2^t\rangle_{t+1}\right)\right)\end{array}\right)$	173, 174	18
173	$t\in\mathbb{N}^+\vdash \left(\sum^{(2\cdot 2^t)_{k=0}}{}^{-1}\left(\text{和}^{2\pi i\varphi k}\lvert k\rangle_{t+1}\right)\right) = \left(\begin{array}{c}\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\pi i\varphi\cdot k}\lvert k\rangle_{t+1}\right)\right)+ \\ \left(\sum_{k=2^t}^{(2\cdot 2^t)-1}\left(\mathsf{e}^{2\pi i\varphi\cdot k}\lvert k\rangle_{t+1}\right)\right)\end{array}\right)$	求和拆分	17
174	$t\in\mathbb{N}^+\vdash \left(\sum_{k=2^t}^{(2\cdot 2^t)-1}\left(\mathsf{e}^{2\pi i\varphi k}\lvert k\rangle_{t+1}\right)\right) = \left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\pi i\varphi\left(k+2^t\right)}\lvert k+2^t\rangle_{t+1}\right)\right)$	求和移位	18
178	$t\in\mathbb{N}^+\vdash \begin{array}{l}\left(\sum_{k=0}^{2^t-1}\left(\left(\mathsf{e}^{2\cdot\pi\cdot i\cdot\varphi\cdot k}\cdot\mathsf{e}^{2\cdot\pi\cdot i\cdot\varphi\cdot 2^t}\right)\cdot(\lvert 1\rangle\otimes\lvert k\rangle_t)\right)\right) \\ \quad = \left(\mathsf{e}^{2\cdot\pi\cdot i\cdot\varphi\cdot 2^t}\cdot\left(\lvert 1\rangle\otimes\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\cdot\pi\cdot i\cdot\varphi\cdot k}\cdot\lvert k\rangle_t\right)\right)\right)\right)\end{array}$	线性性 张量乘积	27
181	$t\in\mathbb{N}^+\vdash \left(\lvert 0\rangle\otimes\left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\pi i\varphi k}\lvert k\rangle_t\right)\right)\right) = \left(\sum_{k=0}^{2^t-1}\left(\mathsf{e}^{2\pi i\varphi k}\left(\lvert 0\rangle\otimes\lvert k\rangle_t\right)\right)\right)$	线性性 张量乘积	30
218	$t\in\mathbb{N}^+,\ \ k\in\{0\ldots 2^t-1\}\vdash \lvert k\rangle_{t+1} = (\lvert 0\rangle\otimes\lvert k\rangle_t)$	ket 的定义	29
350	$t\in\mathbb{N}^+,\ \ k\in\{0\ldots 2^t-1\}\vdash \lvert 2^t+k\rangle_{t+1} = (\lvert 1\rangle\otimes\lvert k\rangle_t)$	ket 定义	24

TABLE V: A 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 Fig. 15 leading to the formal proof.

pp. 124–134.

[12] D. Boneh and R. J. Lipton, Quantum cryptanalysis of hidden linear functions, in *Advances in Cryptology — CRYPT0' 95* (Springer Berlin Heidelberg, 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 Computational Molecular Science **12**, e1580 (2022), available at `https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.1580`.

[15] C. Chareton, S. Bardin, D. Lee, B. Valiron, R. Vilmart, and Z. Xu, Formal methods for quantum programs: A survey (2022), available at `https://arxiv.org/abs/2109.06493`.

[16] K. Hietala, R. Rand, S.-H. Hung, X. Wu, and M. Hicks, Verified optimization in a quantum intermediate representation (2019), arXiv:1904.06319 [cs.LO].

[17] K. Hietala, R. Rand, S.-H. Hung, X. Wu, and M. Hicks, A verified optimizer for quantum circuits, Proceedings of the ACM on Programming Languages **5**, 1 (2021).

[18] J. Paykin, R. Rand, and S. Zdancewic, Qwire: A core

language for quantum circuits, in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017 (Association for Computing Machinery, New York, NY, USA, 2017) p. 846–858.

[19] R. Rand, J. Paykin, and S. Zdancewic, Phantom types for quantum programs (2018), available at https://arxiv.org/abs/2204.07112.

[20] R. Rand, J. Paykin, and S. Zdancewic, QWIRE practice: Formal verification of quantum circuits in coq, Electronic Proceedings in Theoretical Computer Science **266**, 119 (2018).

[21] T. C. D. Team, The coq proof assistant (2022), available at https://zenodo.org/record/7313584#.Y36kuOzMKu4.

[22] The Coq Proof Assistant, (n.d.), last accessed on 11/21/2022 at https://coq.inria.fr/.

[23] C. Chareton, S. Bardin, F. Bobot, V. Perrelle, and B. Valiron, An automated deductive verification framework for circuit-building quantum programs., in *Programming Languages and Systems : 30th European Symposium on Programming (ESOP 2021)*, Lecture Notes in Computer Science. 12648, Vol. 12648 (Springer International Publishing, 2021) pp. 12648:148–177, available at https://link.springer.com/chapter/10.1007/978-3-030-72019-3_6.

[24] Qbricks, Qbricks (n.d.), https://qbricks.github.io/. Last accessed on 2/01/2023.

[25] T. Liu, Y. Li, S. Wang, M. Ying, and N. Zhan, A theorem prover for quantum Hoare logic and its applications (2016), arXiv:1601.03835 [cs.LO].

[26] J. Liu, B. Zhan, S. Wang, S. Ying, T. Liu, Y. Li, M. Ying, and N. Zhan, Formal verification of quantum algorithms using quantum Hoare logic, in *Computer Aided Verification*, edited by I. Dillig and S. Tasiran (Springer International Publishing, 2019) pp. 187–207.

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

[28] M. Ying, Floyd-hoare logic for quantum programs., ACM Transactions on Programming Languages & Systems **33**, 19 (2011).

[29] M. Ying, Toward automatic verification of quantum programs, Formal Aspects of Computing **31**, 3 (2018).

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

[31] 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 (2020), arXiv:2012.10987 [cs.LO].

[32] Prove-It, Welcome to the expanding Prove-It library of proofs (2023), www.pyproveit.org.

[33] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, Quantum algorithms revisited, Proceedings: Mathematical, Physical and Engineering Sciences **454**, 339 (1998), stable URL: https://www.jstor.org/stable/53169.

[34] N. D. Mermin, *Quantum computer science : an introduction.* (Cambridge University Press, 2007).

[35] G. Benenti, G. Casati, and G. Strini, *Principles of Quantum Computation and Information*, Principles of Quantum Computation and Information No. v. 1 (World Scientific, 2004).

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

[37] R. E. Hodel, *An Introduction to Mathematical Logic* (Dover Publications, Inc., Mineola, NY, 1995/2020).

[38] E. Mendelson, *Introduction to Mathematical Logic (6th Ed.)*, 6th ed., Textbooks in Mathematics (CRC Press/Taylor & Francis Group, Boca Raton, Fla., 2015).

[39] L. Lamport, How to write a proof, American Mathematical Monthly **102**, 600 (August-September 1995), available at http://lamport.azurewebsites.net/pubs/lamport-how-to-write.pdf.

[40] L. Lamport, How to write a 21st century proof, Journal of Fixed Point Theory and Applications **11**, 43 (2012), available at http://lamport.azurewebsites.net/pubs/proof.pdf.

[41] H. Barendregt and E. Barendsen, Autarkic computations in formal proofs, Journal of Automated Reasoning **28**, 321 (2002).

[42] H. Barendregt and W. Freek, The challenge of computer mathematics, Philosophical Transactions of the Royal Society A **363**, 2351 (2002).

[43] J. R. Shoenfield, *Mathematical Logic*, Addison-Wesley Series in Logic (Addison-Wesley Publishing Co., Reading, MA, 1967).

[44] B. Eastin and S. T. Flammia, Q-circuit tutorial (2004), available at https://arxiv.org/abs/quant-ph/0406003.

[45] N. G. de Bruijn, A survey of the project automath, in *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, edited by J. Seldin and J. Hindley (Academic Press, 1980) pp. 579–606, available at https://pure.tue.nl/ws/files/2092478/597605.pdf.

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

[47] F. Bauer-Marquart, S. Leue, and C. Schilling, symqv: Automated symbolic verification of quantum programs (2022), available at https://arxiv.org/abs/2212.02267.

[48] 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.

[49] C. Lattner and V. Adve, LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation, in *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)* (Palo Alto, California, 2004).

[50] LLVM, The LLVM Compiler Infrastructure project (n.d.), llvm.org.

[51] Why3, (2023), last accessed on 2/01/2023 at https://why3.lri.fr/.

[52] W. Witzel, M. Sarovar, and K. Rudinger, Versatile Formal Methods Applied to Quantum Information (2015), prod.sandia.gov/techlib/access-control.cgi/2015/159617r.pdf.