Quantum Natural Proof

ANONYMOUS AUTHOR(S)

1 QAFNY: A HIGH-LEVEL QUANTUM LANGUAGE ADMITTING A PROOF SYSTEM

We designed Qafny , the core language of QNP, to express quantum programs with high-level operations that are abstracted away low-level circuit gates. The operations in Qafny are analogized to classical array aggregate operations so that automated verification is feasible. Qafny 's type system tracks the transformation of sessions with three types representing the sessions' quantum states. The Qafny proof system captures the analogy of viewing quantum operations as classical aggregate operations by utilizing the type system to ensure the session state forms. All of these features are novel to quantum languages and proof systems. This section presents Qafny states and the language's syntax, typing, semantics, proof system, and soundness/completeness results.

As a running example, we use the Shor's algorithm [49] shown in Figure 1. Given an integer N, Shor's algorithm finds its nontrivial prime factors, which has the following step: (1) randomly pick a number $1 < a < N$ and compute $k = \text{gcd}(a, N)$; (2) if $k \neq 1$, k is the factor; (3) otherwise, a and N are coprime and we find the order p of a and N; **[Yi: Order of "a mod N"?]** (4) if p is even and $a^{\frac{p}{2}} \neq -1\%N$, $\text{gcd}(a^{\frac{p}{2}} \pm 1, N)$ are the factors, otherwise, we repeat the process. Step (2) is the Shor's algorithm's quantum component and Figure 1 and **??** show its automated proof in QNP. In **??**, we show the actual implementation and proof in the Qafny tool. **[Yi: say a bit more about the pre/post-conditions? Intuitively I would've expected to see that $gcd(r, N) \neq 1$, as I thought the output r would be a factor of N. That isn't what the post-condition says. Also, the figure title says Q-Dafny while we probably mean Qafny, right?]** The Shor's pre-measurement quantum steps in Figure 1 can be analogized as an efficient array filter operation in **??**. The steps before line 14 (steps at line 2, 5 and 9-11) create a 2^n-length of pairs, each of which is formed as $(i, a^i\%N)$ where $i \in [0, 2^n)$. The measurement in line 14 filters the array as a new one ($x[0..n]$) with all elements i satisfying $a^i\%N = u$ where u is a randomly picked number. Notice that modulo multiplication $f(i) = a^i\%N$ is a periodic function. All elements in $x[0..n]$ satisfy $a^i\%N = u$, which means that 1) there is a smallest t such that $a^t\%N = u$, and 2) all elements can be rewritten as $i = t + kp$ and p is the period of the modulo multiplication function, which is given as the post-condition on the right of line 15. Notice that only the black and purple parts in Figure 1 are required to input in the Qafny implementation, and the teal parts can be inferred by the Qafny proof system. [1]

We first introduce Qafny states, syntax, and type system. Then, we discuss its semantics and proof system and metatheories.

1.1 Classical Values and Quantum State and Equations

Qafny has three *kinds* [2] of parameters in Figure 2: `C`-kind classical integers [3]; `M`-kind classical integers (r, n) with a probability characteristic r representing the theoretical probability of the measurement resulting in the natural number n; and $(\texttt{Q}\ n)$-kind quantum parameters, where n represents the number of qubits in a qubit array piece.

Quantum parameters represent qubit array pieces that are appeared as parts of virtual quantum arrays and represent consecutive physical qubit array segments. Qafny represents virtual qubit

[1] The purple is also not needed if we verify the whole Shor's algorithm in **??**.
[2] `C` and `M` kinds are also used as context modes in type checking. See Figure 4.
[3] In the Qafny implementation, one can use any classical types in Dafny. For simplicity, we only allow integers in this paper.

```
$\{A(x) * A(y)\}$ where $A(\beta) = \beta[0..n] \mapsto |\overline{0}\rangle$                 $\{x[0..n] : \mathsf{Nor}, y[0..n] : \mathsf{Nor}\}$
$x \leftarrow \mathsf{H}$;
$\{x[0..n] \mapsto \|\mathsf{H}\|(|\overline{0}\rangle) * A(y)\}$                               $\{x[0..n] : \mathsf{Had}, y[0..n] : \mathsf{Nor}\}$
$\Rightarrow \{x[0..n] \mapsto C * A(y)\}$ where $C = \frac{1}{\sqrt{2^n}} \bigotimes_{j=0}^{n} (|0\rangle + |1\rangle)$    $\{x[0..n] : \mathsf{Had}, y[0..n] : \mathsf{Nor}\}$
$y \leftarrow y{+}1$;
$\{x[0..n] \mapsto C * y[0..n] \mapsto \|y{+}1\|(|\overline{0}\rangle)\}$                  $\{x[0..n] : \mathsf{Had}, y[0..n] : \mathsf{Nor}\}$
$\Rightarrow \{x[0..n] \mapsto C * y[0..n] \mapsto |\overline{0}\rangle\,|1\rangle\}$                 $\{x[0..n] : \mathsf{Had}, y[0..n] : \mathsf{Nor}\}$
$\Rightarrow \{E(0)\}$ where $E(t) =$                                      $\{x[0..n] : \mathsf{Had}, \{x[0..0], y[0..n]\} : \mathsf{CH}\}$
        $x[t..n] \mapsto \frac{1}{\sqrt{2^{n-t}}} \bigotimes_{i=0}^{n-t} (|0\rangle + |1\rangle) *$
        $\{x[0..t], y[0..n]\} \mapsto \Sigma_{i=0}^{2^t} \frac{1}{\sqrt{2^t}} |i\rangle\,|a^i \,\%\, N\rangle$
for (int $j$ := 0; $j < n$ && $x[j]$ ; ++$j$)
    $\{E(j)\}$                                                      $\{x[j..n] : \mathsf{Had}, \{x[0..j], y[0..n]\} : \mathsf{CH}\}$
    $y \leftarrow a^{2^j}\, y \,\%\, N$;
$\{E(n)\}$                                                          $\{x[0..0] : \mathsf{Had}, \{x[0..n], y[0..n]\} : \mathsf{CH}\}$
$\Rightarrow \{\{x[0..n], y[0..n]\} \mapsto \Sigma_{i=0}^{2^n} \frac{1}{\sqrt{2^n}} |i\rangle\,|a^i \,\%\, N\rangle\}$   $\{\{x[0..n], y[0..n]\} : \mathsf{CH}\}$
let $u$ = measure($y$) in ...
$\left\{ \begin{array}{l} x[0..n] \mapsto \frac{1}{\sqrt{s}} \Sigma_{k=0}^{s} |t + kp\rangle \wedge p = \mathsf{ord}(a, N) \\ \wedge\, u = (\frac{p}{2^n}, a^t \,\%\, N) \wedge s = \mathsf{rnd}(\frac{2^n}{p}) \end{array} \right\}$     $\{\{x[0..n]\} : \mathsf{CH}\}$
```

Fig. 1. Pre-measurement quantum steps of the Shor's algorithm. Type environments are on the right. $\mathsf{ord}(a, N)$ gets the order of a and N. $\mathsf{rnd}(r)$ rounds r to the nearest integer. The right-hand-side contains the types for the sessions involved. $|i\rangle$ is an abbreviation of $|(\!|i|\!)\rangle$. $(\!|i|\!)$ turns a number i to a bitstring.

Basic Terms:

Nat. Num	m, n	$\in$	$\mathbb{N}$	Real	r	$\in$	$\mathbb{R}$	Amplitude	z	$\in$	$\mathbb{C}$	Phase	$\alpha(r)$	::=	$e^{2\pi i r}$
Variable	x, y			Bit	d	::=	$0 \mid 1$	Bitstring	c	$\in$	d^+	Basis	β	::=	$(\|c\rangle)^+$

Modes, Kinds, Types, and Classical/Quantum Values:

Mode	g	::=	$\mathsf{C} \mid \mathsf{M}$		
Classical Value	v	::=	$n \mid (r, n)$		
Kind	$\overline{g}$	::=	$g \mid \mathsf{Q}\; n$		
Quantum Type	τ	::=	Nor	$\mid \mathsf{Had}$	$\mid \mathsf{CH}$
Quantum Value	q	::=	$z\beta$	$\mid \frac{1}{\sqrt{2^n}} \bigotimes_{j=0}^{n} (\|0\rangle + \alpha(r_j)\,\|1\rangle)$	$\mid \Sigma_{j=0}^{m} z_j \beta_j$

Quantum Sessions, Environment, and States

Range	re	::=	$x[n..m]$		
Session	κ	::=	$\overline{re}$	concatenated op	$\uplus$
Type Environment	σ	::=	$\overline{\kappa : \tau}$	concatenated op	$\cup$
Quantum State	φ	::=	$\overline{\kappa : q}$	concatenated op	$\cup$

Fig. 2. QAFNY element syntax. Each range $x[n..m]$ in a session l represents the number range $[n, m)$ in a qubit array piece x. Sessions are finite lists, while type environments and states are finite sets. the operations after "concatenated op" refer to the concatenation operations for session, type environments and quantum states.

arrays as *sessions* (κ), which consist of different *disjoint ranges*, each of which describes an array fragment $x[n..m]$, where x is a parameter and $[n..m]$ represents the array fragment from position n to m (exclusive) in array piece x. For simplicity, we assume that there are no aliasing variables in this paper, i.e., two distinct variables represent disjoint array pieces. We view ranges are subsets of session, so that we can abbreviate a singleton session $\{x[n..m]\}$ as a range $x[n..m]$. We can also abbreviate range $x[j, \mathsf{S}\ j]$ to $x[j]$.

Qafny quantum type environments (or states) are finite maps from sessions to types (or state representations), and their domains do not overlap [4]. Quantum state representations are classified as three types: Nor, Had, or CH, representing the three types of state representations in ??. We have subtyping relations over quantum types, i.e., Nor and Had are subtypes of CH, representing that Nor and Had typed quantum values can be rewritten as CH-type state forms.

Qafny utilize *equivalence relations* over quantum sessions, quantum state representations, type environments and states (as shown in ??) to facilitate automated program verification, written as $\equiv$ for state equivalence and $\preceq$ for environment partial order. One example is the state predicate rewrite from line 12 to line 13 in Figure 1, where the session state $x[0..0]$ rewrites to true, because range $x[0..0]$ is essentially empty. The common equivalence relations are state type rewrites, permutations, split and joins. State type rewrites transform state forms, such as the rewrite of session $\{x[0..0], y[0..n]\}$ from type Nor to CH in line 7 and 8 in Figure 1. Another example is to rewrite a CH-type state $\Sigma_{j=0}^{1} z_j\beta_j$ to Nor-type $z_0\beta_0$.

Permutation equivalence refers to two qubits can mutate their locations. One possible permutation can happen in line 13 for write a session $\{x[0..0], y[0..n]\}$ to the form $\{y[0..n], x[0..0]\}$, as:

$$\begin{array}{lcl} \{x[0..0], y[0..n]\} \mapsto \mathsf{CH} & \preceq & \{y[0..n], x[0..0]\} \mapsto \mathsf{CH} \\ \{x[0..n], y[0..n]\} \mapsto \Sigma_{i=0}^{2^n} \frac{1}{\sqrt{2^n}} \|i\rangle\,|a^i \,\%\, N\rangle & \equiv & \{y[0..n], x[0..n]\} \mapsto \Sigma_{i=0}^{2^n} \frac{1}{\sqrt{2^n}} |a^i \,\%\, N\rangle\,|i\rangle \end{array}$$

Notice that the domains of a state and type environment are required to be the same in the Qafny proof system, as the above example. State joins merge two sessions together. Merging a Nor-type and CH-type state is analogized to concatenate two qubit arrays. An example and its explanation are given in ?? line 4-6, where the Nor-type qubit $x[\mathsf{S}\ j]$ is added to the end of CH-type session $x[0..\mathsf{S}\ j]$. Merging a Had-type and CH-type state doubles the CH-type basis states. In each loop step in Figure 1 line 9-11, we add a Had type qubit $x[j]$ to the middle of a CH type session $\{x[0..j], y[0..n]\}$, and the state becomes:

$$\{x[0..\mathsf{S}\ j], y[0..n]\} \mapsto \Sigma_{i=0}^{2^j} \frac{1}{\sqrt{2^j}} |i\rangle\,|0\rangle\,|a^i \,\%\, N\rangle + \Sigma_{j=0}^{2^j} \frac{1}{\sqrt{2^j}} |i\rangle\,|1\rangle\,|a^i \,\%\, N\rangle$$

Notice that the basis states are still all distinct because the two parts above are distinguished by the $|0\rangle$ and $|1\rangle$ of the $x[j]$ position. Merging two CH-type states computes the Cartesian product of basis states in the two sessions (??). State split cuts a session into two individual sessions. The split of Nor and Had types is no more than an array split, while the split of a CH-type is equal to disentanglement, a hard problem. In quantum algorithms, splitting Nor and Had types are more common than disentanglement, and is permitted in Qafny . For splitting CH-type, we invent an upgraded type system in ?? to permit few cases.

1.2 Syntax and Type System

One key Qafny principle permits programmers thinking of quantum programs as sequences of functional operations that are analogized to array aggregate operations, reflected by the Qafny syntax in Figure 3. A program consists of a sequence of C-like statements s that end at a skip ({}). The let operation (let x = e in s) in the first row binds a new classical variable x with e, which is used in s. If e is an arithmetic expression (a), it introduces a C/M kind classical variable. let x = measure(y) in s measures array piece y, stores the result in M-kind variable x that is used in s. The last three of the first row are quantum data-flow operations. $l \leftarrow op$ prepares a quantum superposition state of quantum qubits l through Hadamard H or QFT gates, which are used for state preparation. It is also used to Fourier transform quantum qubit states by a QFT^{-1} gate in quantum phase estimation and

[4]For all $\kappa, \kappa' \in \mathsf{dom}(\sigma)$ (or $\mathsf{dom}(\varphi)$), $\kappa \neq \kappa' \Rightarrow \kappa \cap \kappa' = \emptyset$.

$$
\begin{array}{llcl}
\text{OQASM Expr} & \mu & & \\
\text{Parameter} & l & ::= & x \mid x[a] \\
\text{Arith Expr} & a & ::= & x \mid v \mid a + a \mid a * a \mid \dots \\
\text{Bool Expr} & b & ::= & x[a] \mid (a = a)@x[a] \mid (a < a)@x[a] \mid \dots \\
\text{Predicate} & P, Q, R & ::= & a = a \mid a < a \mid \kappa \mapsto q \mid P \wedge P \mid P * P \mid \dots \\
\text{Gate Expr} & op & ::= & \mathtt{H} \mid \mathtt{QFT}^{[-1]} \\
\text{C/M Mode Expr} & e & ::= & a \mid \mathtt{measure}(y) \\
\text{Statement} & s & ::= & \{\} \mid \mathtt{let}\ x\ =\ e\ \mathtt{in}\, s \mid l \leftarrow op \mid \kappa \leftarrow \mu \mid l \leftarrow \mathtt{dis} \\
 & & \mid & s\ ;\ s \mid \mathtt{if}\ (b)\ s \mid \mathtt{for}\ (\mathtt{int}\ j \in [a_1, a_2)\ \&\&\ b)\ s
\end{array}
$$

Fig. 3. Core QAFNY syntax. OQASM is in Section 1. For an operator OP, $\mathtt{OP}^{[-1]}$ indicates that the operator has a built-in inverse available. $x[a]$ represents the a-th element in the qubit array x, while a quantum variable x represents range $x[0..n]$ and n is the length of x.

$$
\textsc{TPar}\quad \frac{\sigma \preceq \sigma' \qquad \Omega; \sigma' \vdash_g s \triangleright \sigma''}{\Omega; \sigma \vdash_g s \triangleright \sigma''}
\qquad
\textsc{TExp}\quad \frac{x \notin \Omega \qquad \Omega[x \mapsto \mathtt{C}]; \sigma \vdash_g s[n/x] \triangleright \sigma'}{\Omega; \sigma \vdash_g \mathtt{let}\ x\ =\ n\ \mathtt{in}\, s \triangleright \sigma'}
$$

$$
\textsc{TMea}\quad \frac{x \notin \Omega \qquad \sigma(y) = \{y[0..j] \uplus \kappa \mapsto \tau\} \qquad \Omega(y) = \mathtt{Q}\ j \qquad \Omega[x \mapsto \mathtt{M}]; \sigma[\kappa \mapsto \mathtt{CH}] \vdash_{\mathtt{C}} s \triangleright \sigma'}{\Omega; \sigma \vdash_{\mathtt{C}} \mathtt{let}\ x\ =\ \mathtt{measure}(y)\ \mathtt{in}\, s \triangleright \sigma'}
$$

$$
\textsc{TA-CH}\quad \frac{FV(\Omega, \mu) = \kappa \qquad \sigma(\kappa \uplus \kappa') = \mathtt{CH}}{\Omega; \sigma \vdash_g \kappa \leftarrow \mu \triangleright \{\kappa \uplus \kappa' : \mathtt{CH}\}}
\qquad
\textsc{TDis}\quad \frac{FV(\Omega, l) = \kappa \qquad \sigma(\kappa \uplus \kappa') = \mathtt{CH}}{\Omega, \sigma \vdash_g \kappa \leftarrow \mathtt{dis} \triangleright \{l \uplus \kappa' : \mathtt{CH}\}}
$$

$$
\textsc{TSeq}\quad \frac{\Omega; \sigma \vdash_g s_1 \triangleright \sigma_1 \qquad \Omega; \sigma[\uparrow \sigma_1] \vdash_g s_2 \triangleright \sigma_2}{\Omega; \sigma \vdash_g s_1\ ;\ s_2 \triangleright\ \sigma_2 \cup \sigma_1|_{\notin \mathtt{dom}(\sigma_2)}}
$$

$$
\textsc{TIF}\quad \frac{FV(\Omega, b) = \kappa \qquad \kappa \cap FV(\Omega, s) = \emptyset \qquad \Omega; \sigma[\kappa \mapsto \mathtt{CH}] \vdash_{\mathtt{M}} s \triangleright \{\kappa' : \mathtt{CH}\}}{\Omega; \sigma[\kappa \uplus \kappa' \mapsto \mathtt{CH}] \vdash_g \mathtt{if}\ (b)\ s \triangleright \{\kappa \uplus \kappa' : \mathtt{CH}\}}
$$

$$
\textsc{TLoop}\quad \frac{\forall j \in [n_1, n_2)\ .\ \ \Omega[x \mapsto \mathtt{C}]; \sigma[\uparrow \sigma'[j/x]] \vdash_g \mathtt{if}\ (b[j/x])\ s[j/x] \triangleright \sigma'[\mathtt{S}\ j/x]}{\Omega; \sigma[\uparrow \sigma'[n_1/x]] \vdash_g \mathtt{for}\ (\mathtt{int}\ x \in [n_1, n_2)\ \&\&\ b)\ s \triangleright \sigma'[n_2/x]}
$$

$$
\sigma[\uparrow \sigma'] = \sigma[\forall \kappa : \tau \in \sigma'\ .\ \kappa \mapsto \tau] \qquad \sigma|_{\notin \mathtt{dom}(\sigma')} = \{\kappa : \tau \in \sigma | \kappa \notin \mathtt{dom}(\sigma')\}
$$

Fig. 4. QAFNY type system. $\sigma(y) = \{\kappa \mapsto \tau\}$ produces the entry $\kappa \mapsto \tau$ and the range $y[0..|y|] \subseteq \kappa$. $\sigma(\kappa) = \tau$ is an abbreviation of $\sigma(\kappa) = \{\kappa \mapsto \tau\}$. $\sigma[\kappa \mapsto \tau]$ is valid only if $\kappa \in \mathtt{dom}(\sigma)$, we can use rule TPAR to address the domain. $FV(\Omega, -)$ produces a session by union all qubits in $-$ with info in Ω; see ??.

Shor's algorithms. **[Yi: It seems somewhat limiting that we only have Hadamard and QFT for *op*. Or I guess we have Pauli X too by just doing a $+1$, right? On a somewhat unrelated topic, this already seem enough to implement quantum key distribution too, though I don't know if we care. It's one of those popular algorithms that people like to talk about, but the analysis seems trivial (and mostly classical) if we don't have an adversary or a noise model...]** The other gate applications are done through $\kappa \leftarrow \mu$ that performs an OQASM quantum oracle computation μ ([28]) on each basis state of session κ as we unveil in ??. Since all quantum reversible arithmetic operations were defined in OQASM; thus, we permit μ's description in QAFNY to be arithmetic operations, similar to expressions a in Figure 3, e.g., fig. 1 line 5 and 11. $l \leftarrow \mathtt{dis}$ is a quantum diffusion operation applying on the parameter l, where l may be part of an entangled session. The main functionality is to increase and average the occurrence likelihood of some quantum bases in a quantum state.

The second row of statements in Figure 3 are control-flow operations. $s_1\ ;\ s_2$ is a sequential operation. $\mathtt{if}\ (b)\ s$ is a classical or quantum conditional depending on if b contains quantum parameters. Quantum reversible Boolean guards b are implemented as OQASM oracle operations, written as $(a_1 = a_2)@x[a]$, $(a_1 < a_2)@x[a]$, and $x[a]$, meaning that for each quantum basis state, we compute b's value $a_1 = a_2$, $a_1 < a_2$, and `true` [5], and store the result in the qubit bit $x[a]$ by computing $b \oplus x[a]$. One example quantum Boolean equation is used at the quantum walk algorithm in ??. `for` (`int` $j \in [a_1, a_2)$ && b) $\{s\}$ is a possible quantum for-loop depending on if b contains quantum parameters. A classical variable j is introduced and initialized as the lower bound a_1, incremented in each loop step by $++j$, and ended at the upper bound a_2. For example, the for-loop in Figure 1 repeatedly entangles the `Had`-type qubit $x[j]$ with the `CH`-type session $\{x[0..j], y[0..n]\}$ through the modulo multiplication at line 11. [6] **[Yi: I like this.]**

A Quantum Session Type System. In QAFNY , typing is with respect to a *kind environment* Ω and a *finite type environment* σ, which map QAFNY variables to kinds and map sessions to types, respectively. The typing judgment is written as $\Omega; \sigma \vdash_g s \triangleright \sigma'$, which states that statements s is well-typed under the context mode g and environments Ω and σ, the sessions representing s is exactly the domain of σ' ($\mathsf{dom}(\sigma')$), and s transforms types for the sessions in σ to types in σ'. Ω is populated through `let` expressions that introduce variables,and the type system enforces variable scope; such enforcement is neglected in Figure 4 for simplicity. **[Yi: I'm not a PL person, so this is a bit difficult for me to read. The reviewers will probably have an easier time than I did though, so maybe I shouldn't be worried?]** We assume that variables introduced in `let` expressions are all distinct through proper alpha conversions, as in TEXP and TMEA. $\mathsf{dom}(\sigma)$ and $\mathsf{dom}(\sigma')$ are large enough to describe all sessions containing all qubits mentioned in s. Kinds g are reused as context modes (`C` and `M`) for enforcing no quantum information leak in a quantum conditional. Selected type rules are given in Figure 4; the rules not mentioned are similar and listed in ??.

The type system enforces four invariants. First, we place well-formed and context restrictions on quantum programs. Well-formedness refers to the No-cloning theorem, such that qubits mentioned in a quantum conditional Boolean guard cannot be accessed in the conditional body; while context restriction refers to no quantum information leak, such that quantum conditional bodies cannot create and measure (`measure`) qubits. For example, the FV checks in rule TIF enforces that the session for the Boolean and the conditional body does not overlap. Context mode `C` permits most QAFNY operations. Once a type rule turns it to `M`, as in TIF, we disallow measurement operations, as rules TMEA is valid only if the input context mode is `C`. Second, the type system tracks the arrangement of sessions as well as permits the session equivalence relations through rule TPAR. In rule TA-CH, the session appearing in μ is κ, which is the prefix of a session $\kappa \uplus \kappa'$. To type check the case when we apply μ on other locations, we utilize the TPAR to transform the session structure by the session permutation equivalence. For example, in ?? line 5, we want to apply addition on $x[\mathsf{S}\ j]$, but the session is arranged as $x[0..j+2]$. In type checking the statement, we first rewrite the session through rule TPAR to $\{x[\mathsf{S}\ j..j+2], x[0..\mathsf{S}\ j]\}$, then apply the TA-CH rule. Third, the type system enforces that `C` classical variables can be evaluated to values in the compilation time [7], while tracks `M` variables. Rule TEXP enforces that a classical variable x is replaced with its assignment value n in s, where classical expressions containing x are evaluated. See ??. Finally, the type system estimates the scopes of entangled sessions. In rule TIF, we use κ', the domain of the post type environment, as the estimated session that is large enough to describe all possible qubits directly and indirectly mentioned in s [8] and should be in the same session. Based on κ', we estimate the conditional's session as $\kappa \uplus \kappa'$. The type system keeps such estimation as small as possible. **[Yi: I**

[5] a_1 and a_2 can possibly be a quantum array piece x in an entangled session.

[6] In the QAFNY implementation, $++j$ and $j < a_2$ can be arbitrary monotonic increment and comparison functions. For simplicity, we restrict the two to be $++j$ and $j < a_2$ in this paper.

[7] We consider all computation that only needs classical computer is done in the compilation time.

[8] This includes all entangled array pieces whose qubits are mentioned in s.

$$\textsc{Frame}\quad \frac{FV(s)\cap FV(R)=\emptyset \quad FV(s)\subseteq \mathtt{dom}(\sigma) \quad \sigma\perp\sigma' \quad \Omega;\sigma\vdash_g \{P\}\,s\,\{Q\}}{\Omega;\sigma\cup\sigma'\vdash_g \{P*R\}\,s\,\{Q*R\}} \qquad \frac{\sigma\perp\sigma' \quad \varphi\perp\varphi' \quad \Omega;\sigma;\psi;\varphi\models_g P \quad \Omega;\sigma';\psi;\varphi'\models_g Q}{\Omega;\sigma\cup\sigma';\psi;\varphi\cup\varphi'\models_g P*Q}$$

$$\textsc{TCon}\quad \frac{\sigma\le\sigma' \quad \Omega;\sigma'\vdash_g \{P'\}\,s\,\{Q'\}}{\Omega;\sigma\vdash_g \{P\}\,s\,\{Q\}} \qquad \frac{\forall j.\,|\kappa|=|c_j|}{\Omega;\sigma;\psi;\varphi[\kappa\mapsto \sum_{j=0}^{m} z_j\,|c_j\rangle\,\beta'_j)]\models_g \sum_{j=0}^{m} z_j\,|c_j\rangle\,\beta_j)}$$

$$TC(\sigma,P,Q)\triangleq \Omega;\sigma\vdash_g s\triangleright\sigma_1 \wedge \Omega;\sigma\vdash P\wedge \Omega;\sigma[\uparrow\sigma_1]\vdash Q$$

Fig. 5. Frame proof rule and type consequence rule and model rules

don't understand. Does this just contain the qubit used in the conditional plus the qubits used by the loop body? Why or why not?]

1.3 The Qafny Semantics and Proof System

QNP intends to create a proof system that utilizes classical automated reasoning infrastructure in analyzing quantum programs, through prototyping classical separation logic [44], by analogizing quantum operations to array aggregate operations. To materialize such analogy, Qafny sessions play two roles, both as variable names representing quantum arrays and as qubit array structure indicators, through the enforcement of well-formedness and well-typed restrictions on states and proof system predicates. **??** describes the well-formedness: well-formed domains ($\Omega\vdash\mathtt{dom}(\sigma)$) mean that every range variables in a session in $\mathtt{dom}(\sigma)$ are mentioned in Ω, and well-formed predicates ($\Omega,\sigma\vdash P$) mean that every sessions mentioned in P are in $\mathtt{dom}(\sigma)$. Qafny semantics is a small-step transition $(\psi,\varphi,s)\longrightarrow(\psi',\varphi',s')$, where ψ and ψ' are stacks storing local classical variables, and φ and φ' are quantum states. A Qafny proof triple is written as: $\Omega;\sigma\vdash_g \{P\}\,s\,\{Q\}$, where P and Q are the pre- and post-conditions, Ω, σ, and g are the type entities. Apparently, any QNP proof judgment, which merges the Qafny type system into proof rules, has a hidden type restriction stated as predicate TC in Figure 5. The restriction is required not only at the bottom proof tree, but also at all levels, e.g., rule TCon describes the typing consequence rule by replacing σ with σ'. We require that P and Q satisfy $TC(\sigma,P,Q)$, as well as P' and Q' satisfy $TC(\sigma',P',Q')$. Similarly, the Frame rule is no more than a separation logic frame rule [9], other than the additional type restrictions, where we separate the type environment into σ and σ'; on the above level of the Frame rule, the conditions P and Q also need to satisfy the type restriction with respect to Ω and σ.

The rule besides rule Frame in Figure 5 is the modeling rule for a separable state P and Q. In QNP, a modeling rule has the judgment: $\Omega;\sigma;\psi;\varphi\models_g P$, with the above well-formed restrictions, additionally requiring $\mathtt{dom}(\psi)\subseteq\mathtt{dom}(\Omega)$, and $\Omega;\sigma\vdash_g\varphi$, defined as follows:

Definition 1.1 (Well-formed Qafny state). A state φ is *well-formed*, written as $\Omega;\sigma\vdash_g\varphi$, iff $\mathtt{dom}(\sigma)=\mathtt{dom}(\varphi)$, $\Omega\vdash\mathtt{dom}(\sigma)$, and:

- For every $\kappa\in\mathtt{dom}(\sigma)$ such that $\sigma(\kappa)=\mathtt{Nor}$, $\varphi(\kappa)=z\,|c\rangle$ and $|\kappa|\le|c|$ [10] and $|z|\le 1$; specifically, if $g=\mathtt{C}$, $|\kappa|=|c|$ and $|z|=1$. **[Yi: Why $|\kappa|\le|c|$? What is this trying to do?]**
- For every $\kappa\in\mathtt{dom}(\sigma)$ such that $\sigma(\kappa)=\mathtt{Had}$, $\varphi(\kappa)=\frac{1}{\sqrt{2^n}}\bigotimes_{j=0}^{n}(|0\rangle+\alpha(r_j)\,|1\rangle)$ and $|\kappa|=n$.

[9]We use $FV(s)\cap FV(R)=\emptyset$ here because $FV(s)$ produces the session that s modifies in Qafny .

[10]$|\kappa|$ and $|c|$ are the lengths of κ and c.

SA-CH
$$\frac{\varphi(\kappa) = \{\kappa \uplus \kappa' \mapsto \sum_{j=0}^{m} q(c_j)\} \qquad \forall j.\ |c_j| = |\kappa|}{(\varphi, \kappa \leftarrow \mu) \longrightarrow (\varphi[\kappa \uplus \kappa' \mapsto \sum_{j=0}^{m} q([\![\mu]\!](c_j))], \{\})}$$

PA-CH
$$\frac{\sigma(\kappa) = \{\kappa \uplus \kappa' \mapsto \mathtt{CH}\} \qquad \forall j.\ |c_j| = |\kappa|}{\Omega; \sigma \vdash_g \{\kappa \uplus \kappa' \mapsto \sum_{j=0}^{m} q(c_j)\}\ \kappa \leftarrow \mu\ \{\kappa \uplus \kappa' \mapsto \sum_{j=0}^{m} q([\![\mu]\!](c_j))\}}$$

$$q(c_j) = \Sigma_{j=0}^{m} z_j\, |c_j\rangle\, \beta_j \qquad [\![\mu]\!]c_j = z_j'\, |c_j'\rangle$$

Fig. 6. Oracle application rules

- For every $\kappa \in \mathsf{dom}(\sigma)$ such that $\sigma(\kappa) = \mathtt{CH}$, $\varphi(\kappa) = \Sigma_{j=0}^{m} z_j\, |c_j\rangle$ and $|\kappa| \le |c_j|$ and $\sum_{j=0}^{m} |z|^2 \le 1$ and for $i, k \in [0, m)$, $|c_i| = |c_k|$; specifically, if $g = \mathtt{C}$, $|\kappa| = |c_j|$ and $\sum_{j=0}^{m} |z|^2 = 1$.

The reason that we place a relaxed well-formedness for a M-mode state is that the program location in M-mode is inside a quantum conditional where parts of the sessions and states are frozen, whereas once the quantum conditional finishes execution and the program is transitioned back to C-mode, the frozen parts are added back to the state and the united state is required to satisfy the quantum state principles, which are described as the C-mode restrictions in Definition 1.1. The frozen states are described in Section 1.3. There are other rules in ?? that are similar the rules in a separation logic framework with additional QAFNY type restrictions. We now discuss few interesting cases.

Oracle Applications. The oracle application $\kappa \leftarrow \mu$ is analogized to classical array map operation as discussed in ??. Assume that μ works on session $\kappa \uplus \kappa'$. For each element $z_j\, |c_j\rangle\, \beta_j$ in the CH type state, we first find c_j as the corresponding basis state bitstring for the κ session fragment [11], then we apply μ only on c_j, which is described in the semantic rule SA-CH. Rule PA-CH describes the proof rule for capturing the array map analogy, which has the exact behavior as the semantic rule. The other similar rules, such as state preparation rules, can be found in ??.

Rules for Conditionals and For-Loops. Figure 7 describes the analogy of quantum conditionals in QAFNY , which are partial map functions that only apply applications on the red parts and freeze the black parts. It contains two levels of freezing. For each basis state in a session $\kappa_b \uplus \kappa_a$, it freezes the κ_b part of the state, which is indicated as the marked black items in the second line in Figure 7.

For each basis state in $\Sigma_{j=0}^{m} z_j\, |c_j\rangle\, \beta_j$ with $|c_j| = |\kappa_b|$, we freeze the c_j part by pushing it to the end of the basis state as $z_j \beta_j |c_j\rangle$, which is indicated in rule SIF (Figure 10). During the process, the state session is transformed from $\kappa_b \uplus \kappa_a$ to κ_a when evaluating the conditional body s. This indicates that the $\kappa_b / |c_j\rangle$ part is frozen, because the session is shorter, the $|c_j\rangle$ part is stored in a location beyond the reachable of session κ_a; therefore, in evaluating s, no operation can touch the $|c_j\rangle$ part. The second level of freezing happens in selecting valid basis states by evaluating the Boolean condition b on basis state bitstrings. Predicate R in ruleSIF reflects the task. Here, we divide the state into two parts as $\Sigma_{j=0}^{m} z_j\, |c_j\rangle\, \beta_j$ and $q(\kappa, \neg b)$, where the first part contains all basis states satisfying b, i.e., if we replace the qubit variables in b with c_j, the evaluation $[\![b[c_j/\kappa]]\!]$ is true; while the second part $q(\kappa, \neg b)$ contains all basis states that evaluate b to false. After the freezing step, we execute s only on the selected unfrozen parts of the state, and merge the execution

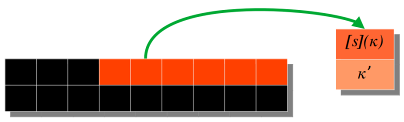

Fig. 8. Conditional Analogy

[11] κ and c_j have the same length and are the prefix session and basis state, respectively.

$$\frac{R \qquad \Omega;\sigma;\varphi \models_g \kappa \mapsto \sum_{j=0}^{m} z_j\beta_j\,|c_j\rangle}{\Omega;\sigma;\varphi \models_g \mathcal{M}(b,\kappa) \mapsto \sum_{j=0}^{m} z_j\,|c_j\rangle\,\beta_j + q(\kappa,\neg b)}
\qquad
\frac{R \qquad \Omega;\sigma,\varphi \models_g \kappa \mapsto \sum_{j=0}^{m'} z'_j\,|c_j\rangle\,\beta'_j + q(\kappa,\neg b)}{\begin{array}{r}\Omega;\sigma,\varphi \models_g \mathcal{U}(\neg b,\kappa) \mapsto \sum_{j=0}^{m} z_j\,|c_j\rangle\,\beta_j + q(\kappa,\neg b)\\ *\mathcal{U}(b,\kappa) \mapsto \sum_{j=0}^{m'} z'_j\beta'_j\,|c_j\rangle\end{array}}$$

SIF

$$\frac{R \qquad FV(\emptyset,s)\subseteq\kappa' \qquad (\psi,\varphi[\kappa'\mapsto \sum_{j=0}^{m} z_j\beta_j\,|c_j\rangle],s)\longrightarrow(\psi',\varphi[\kappa'\mapsto\sum_{j=0}^{m'} z'_j\beta'_j\,|c_j\rangle],s')}{(\psi,\varphi[\kappa\uplus\kappa'\mapsto\sum_{j=0}^{m} z_j\,|c_j\rangle\,\beta_j+q(\kappa,\neg b)],\texttt{if }(b)\ s)\longrightarrow(\psi',\varphi[\kappa\uplus\kappa'\mapsto\sum_{j=0}^{m'} z'_j\,|c_j\rangle\,\beta'_j+q(\kappa,\neg b)],\texttt{if }(b)\ s')}$$

PIF

$$\frac{\Omega;\{\kappa':\texttt{CH}\}\vdash Q' \qquad \Omega;\sigma[\kappa'\mapsto\texttt{CH}]\vdash_{\mathsf{M}}\{P[\mathcal{M}(b,\kappa')/\kappa\uplus\kappa']\}\ s\ \{Q*Q'\}}{\Omega;\sigma[\kappa\uplus\kappa'\mapsto\texttt{CH}]\vdash_g\{P\}\ \texttt{if }(b)\ s\ \{P[\mathcal{U}(\neg b,\kappa\uplus\kappa')/\kappa\uplus\kappa')]*Q'[\mathcal{U}(b,\kappa\uplus\kappa')/\kappa']\}}$$

SLOOP

$$\frac{n_1<n_2 \qquad b'=b[n_1/j] \qquad s'=s[n_1/j]}{\begin{array}{l}(\psi,\varphi,\texttt{for (int }j\in[n_1,n_2)\ \&\&\ b)\ s)\longrightarrow\\ \quad(\psi,\varphi,\texttt{if }(b')\ s'\ ;\ \texttt{for (int }j\in[\mathsf{S}\ n_1,n_2)\ \&\&\ b)\ s)\end{array}}$$

PLOOP

$$\frac{n_1<n_2 \qquad \Omega;\sigma\vdash_{\mathsf{M}}\{P(j)\wedge j<n_2\}\ \texttt{if }(b)\ s\ \{P(\mathsf{S}\ j)\}}{\Omega;\sigma\vdash_g\{P(n_1)\}\ \texttt{for (int }j\in[n_1,n_2)\ \&\&\ b)\ s\ \{P(n_2)\}}$$

$$q(\kappa,\neg b)=\Sigma_{i=0}^{m} z_i\,|c_i\rangle\,\beta_i \ \ \texttt{where}\ \forall i.\ |c_i|=|\kappa|\wedge[\![\neg b[c_i/\kappa]]\!]$$
$$R=FV(\emptyset,b)=\kappa\wedge\forall j.\ |c_j|=|\kappa|\wedge[\![b[c_j/\kappa]]\!]$$

Fig. 7. Semantic and Proof Rules for Conditionals and For-loops. $\mathcal{M}$ is the frozen function and $\mathcal{U}$ is the unfrozen function. $[\![b[c_j/\kappa]]\!]$ is the interpretation of Boolean guard b by replacing qubits mentioned in κ with bits in bitstring c_j. $P(j)$ is a predicate P with j as a variable.

results (z'_j and β'_j) back to the whole state. The result state element number m' might be different from the pre-execution one (m) because applications in s might increase the state numbers such as applying a quantum diffusion operation.

To design a proof rule for such partial map, we develop the frozen ($\mathcal{M}$) and unfrozen ($\mathcal{U}$) operations with the session types. Both take a Boolean expression b and a quantum state as arguments. As shown on top of Figure 10, $\mathcal{M}$'s modeling materializes the freezing mechanism above. For a state $\Sigma_{j=0}^{m} z_j\,|c_j\rangle\,\beta_j+q(\kappa,\neg b)$, we only keep basis states satisfying b, as shown in predicate R, remove the unsatisfied basis states $q(\kappa,\neg b)$, and push the basis bitstrings c_j to the ends of the basis states, which are similar to stacks that store frozen bitstrings. In the pre-condition manipulation of rule PIF, we substitute $\kappa\uplus\kappa'$ with $\mathcal{M}(b,\kappa')$. During the process, the session type in σ is changed from $\kappa\uplus\kappa'$ to κ'. Function $\mathcal{U}$ unfreezes the state by assembling the result state, of applying s, back to the frozen $\kappa\uplus\kappa'$ state (the black part in Figure 7). It is always appeared as a pair with both the b and $\neg b$ cases, referring to the two state divisions above. In the post-condition manipulation, we substitute $\kappa\uplus\kappa'$ with $\mathcal{U}(\neg b,\kappa\uplus\kappa')$ in P, representing the unchanged and frozen state; and substitute κ' with $\mathcal{U}(b,\kappa\uplus\kappa')$ in Q', representing the result of applying s on the unfrozen part, and assemble them together through the separation operation $*$. $\Omega;\{\kappa':\texttt{CH}\}\vdash Q'$ ensures that Q' only mentions session κ'. $\mathcal{U}$'s modeling in the first line of Figure 10 captures the assemble procedure by merging two $\mathcal{U}$ constructs together. Notice that c_j appears in two $\mathcal{U}$ constructs, meaning that we actually utilize c_j to distinguish the frozen and unfrozen basis states without using b. Rules SLOOP and

$$\dfrac{|c| = n \qquad \Omega;\sigma;\varphi \models \kappa \mapsto \sum_{j=0}^{m} \frac{z_j}{\sqrt{r}} |c_j\rangle \wedge x = (r, \{\!|c|\!\})}{\Omega;\sigma,\varphi \models \mathcal{F}(x,n,\kappa) \mapsto \sum_{j=0}^{m} z_j\, |c\rangle\, |c_j\rangle + q(n, \neq c)}$$

$$\textsc{SMea}\quad \dfrac{\varphi(y) = \{y[0..n] \uplus \kappa \mapsto \sum_{j=0}^{m} z_j\, |c\rangle\, |c_j\rangle + q(n, \neq c)\}}{\begin{array}{c}(\psi, \varphi, \texttt{let } x \;=\; \texttt{measure}(y) \texttt{ in } s) \longrightarrow \\ (\psi[x \mapsto (r, \{\!|c|\!\})], \varphi[\kappa \mapsto \sum_{j=0}^{m} \frac{z_j}{\sqrt{r}} |c_j\rangle], s)\end{array}}$$

$$\textsc{PMea}\quad \dfrac{\Omega[x \mapsto \mathsf{M}];\sigma[\kappa \mapsto \mathsf{CH}] \vdash_{\mathsf{C}} \{P[\mathcal{F}(x,n,\kappa)/y[0,n] \uplus \kappa]\}\, s\, \{Q\}}{\Omega[y \mapsto \mathsf{Q}\ n];\sigma[y[0,n] \uplus \kappa \mapsto \mathsf{CH}] \vdash_{\mathsf{C}} \{P\}\, \texttt{let } x \;=\; \texttt{measure}(y) \texttt{ in } s\, \{Q\}}$$

$$r = \Sigma_{k=0}^{m} |z_k|^2 \qquad\qquad q(n, \neq c) = \Sigma_{k=0}^{m'} z'_k\, |c'.c'_k\rangle \ \texttt{where } c' \neq c$$

Fig. 9. Semantic and Proof Rules for Measurement. $\mathcal{F}$ is the measurement function construct. $\{\!|c|\!\}$ turns bitstring c to an integer, and r is the likelihood that the bitstring c appears in a basis state.

PLOOP are the semantic and proof rules for a for-loop, which is a simplified version of while-loop in classical separation logic, because for-loop is guaranteed to terminate.

$$\dfrac{\dfrac{\dfrac{\Omega;\{\kappa : \mathsf{CH}\} \vdash_{\mathsf{M}} \{\kappa \mapsto \frac{1}{\sqrt{2}} |\bar{1}\rangle\, |0\rangle |1\rangle\}\, s\, \{\kappa \mapsto \frac{1}{\sqrt{2}} |\bar{1}\rangle\, |1\rangle |1\rangle\}}{\Omega;\{\kappa : \mathsf{CH}\} \vdash_{\mathsf{M}} \{\mathcal{M}(x[j],\kappa) \mapsto \sum_{i=0}^{2} \frac{1}{\sqrt{2}} |\bar{i}\rangle\, |0\rangle\}\, s\, \{\kappa \mapsto \frac{1}{\sqrt{2}} |\bar{1}\rangle\, |1\rangle |1\rangle\}}}{\Omega;\{\kappa_1 : \mathsf{CH}\} \vdash_{\mathsf{C}} \{\kappa_1 \mapsto \sum_{i=0}^{2} \frac{1}{\sqrt{2}} |\bar{i}\rangle\, |0\rangle\}\, \texttt{if } (x[j])\, s\, \{\mathcal{U}(\neg x[j],\kappa_1) \mapsto \sum_{i=0}^{2} \frac{1}{\sqrt{2}} |\bar{i}\rangle\, |0\rangle * \mathcal{U}(x[j],\kappa_1) \mapsto \frac{1}{\sqrt{2}} |\bar{1}\rangle\, |1\rangle |1\rangle\}}}{\Omega;\sigma \vdash_{\mathsf{C}} \{x[0..\mathsf{S}\ j] \mapsto \sum_{i=0}^{2} \frac{1}{\sqrt{2}} |\bar{i}\rangle * x[\mathsf{S}\ j..n] \mapsto |\bar{0}\rangle\}\, \texttt{if } (x[j])\, s\, \{x[0..j+2] \mapsto \sum_{i=0}^{2} \frac{1}{\sqrt{2}} |\bar{i}\rangle * x[j+2..n] \mapsto |\bar{0}\rangle\}}$$

$$\kappa = \{x[0..j], x[\mathsf{S}\ j..j+2]\} \quad \kappa_1 = \{x[j..\mathsf{S}\ j\} \uplus \kappa \quad s = x[\mathsf{S}\ j] \leftarrow x[\mathsf{S}\ j] + 1; \quad \sigma = \{x[0..\mathsf{S}\ j] : \mathsf{CH}, x[\mathsf{S}\ j..n] : \mathsf{Nor}\}$$

As an example, we show the proof, built from bottom up, for a GHZ loop-step above. We first move a Nor type qubit $x[\mathsf{S}\ \ j]$ to session $x[0..\mathsf{S}\ \ j]$ and use the Frame rule to disregard session $x[j+2..n]$. We then apply rule PIF to freeze the qubit $x[j]$, with the basis state having $x[j] = 0$ frozen and disregarded, and left with the state $\frac{1}{\sqrt{2}} |\bar{1}\rangle\, |0\rangle |1\rangle$, with the purple part also being frozen. Notice that κ_1 has the same qubits as session $x[0..j+2]$ with different arrangement by applying some equivalence state rewrites. We apply rule PA-CH at the top and flip the 0 bit. As the post-condition of rule PIF, we use two $\mathcal{U}$ functions to assemble the state $\frac{1}{\sqrt{2}} |\bar{1}\rangle\, |1\rangle |1\rangle$ back to the state of session κ_1.

Measurement Rules. As Figure 8 describes, quantum measurement is a two-step array filter: 1) The session is partitioned into two parts, so do all the basis states, and we randomly pick a basis state first part as a key (the pink part); and 2) we create a new array by cutting all elements' first parts with keeping the elements whose original first part is equal to the key. Notice that the the red basis states appear in a periodical pattern in the whole array. This behavior is universally true for quantum operations, and many quantum algorithms utilize the periodical pattern.

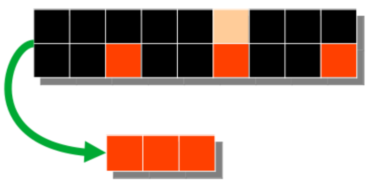
Fig. 10. Measurement Analogy

In rule SMea in Figure 11, we pick an n-length bitstring c as the pink key and collect m basis states $\Sigma_{j=0}^{m} z_j\, |c\rangle\, |c_j\rangle$ that has the key c. In the post-state, we update the remaining session κ to $\Sigma_{j=0}^{m} \frac{z_j}{\sqrt{r}} |c_j\rangle$ with the adjustment of amplitude $\frac{1}{\sqrt{r}}$, and replace the variable x in the statement s with the value $(r, \{\!|c|\!\})$. In designing the proof rule PMea, a session type operation $\mathcal{F}(x,n,\kappa)$ is invented to do exactly the two steps above by selecting an n-length prefix bitstring c in a basis

SDis
$$\frac{FV(\Omega, l) = \kappa \qquad \varphi(\kappa) = \{\kappa \uplus \kappa' \mapsto q\}}{(\varphi, l \leftarrow \texttt{dis}) \longrightarrow (\varphi[\kappa \uplus \kappa' \mapsto \mathcal{D}(|\kappa|, q)], \{\})}$$

PDis
$$\frac{FV(\Omega, l) = \kappa \qquad \sigma(\kappa) = \{\kappa \uplus \kappa : \texttt{CH}\}}{\Omega; \sigma \vdash_g \{\kappa \uplus \kappa' \mapsto q\}\, l \leftarrow \texttt{dis}\, \{\kappa \uplus \kappa' \mapsto \mathcal{D}(|\kappa|, q)\}}$$

$$\mathcal{D}(n, \Sigma_{i=0}^{m} \Sigma_{j=0}^{2^n} z_{ij}\, |j\rangle\, |c_{ij}\rangle) = \Sigma_{i=0}^{m} \Sigma_{j=0}^{2^n} \left(\tfrac{1}{2^{n-1}} \Sigma_{u=0}^{2^n} z_{iu} - z_{ij}\right) |j\rangle\, |c_{ij}\rangle$$

Fig. 11. Semantic and Proof Rules for Diffusion Operations

state for range $y[0..n]$, computing the probability r, and assigning $(r, \{\!|c|\!\})$ to variable x. Rule PMea replaces session $y[0..n] \cup \kappa$ in P with the measurement result session $\mathcal{F}(x, n, \kappa)$ and updates the type state Ω and σ by replacing $y[0..n] \cup \kappa$ with κ.

$$\frac{\Omega[u \mapsto \texttt{M}]; \{x[0..n] : \texttt{CH}\} \vdash_C \{\mathcal{F}(u, n, x[0..n]) \mapsto C\}\, \{\}\, \{x[0..n] \mapsto D * E\}}{\Omega; \{\{y[0..n], x[0..n]\} : \texttt{CH}\} \vdash_C \{\{y[0..n], x[0..n]\} \mapsto C\}\, \texttt{let}\ u\ =\ \texttt{measure}(y)\ \texttt{in}\ \{\}\, \{x[0..n] \mapsto D * E\}}$$

$$C \triangleq \Sigma_{j=0}^{2^n} \tfrac{1}{\sqrt{2^n}}\, |(\!|a^j \ \%\ N|\!).(\!|j|\!)\rangle \quad D \triangleq \tfrac{1}{\sqrt{s}} \Sigma_{k=0}^{s} |t + kp\rangle \quad E \triangleq p = \texttt{ord}(a, N) \wedge u = (\tfrac{s}{2^n}, a^t \ \%\ N) \wedge s = \texttt{rnd}(\tfrac{2^n}{p})$$

Here, we show a proof fragment above for the partial measurement in line 14 in Figure 1. The proof applies rule PMea by replacing session $\{x[0..n], y[0..n]\}$ with $\mathcal{F}(u, n, x[0..n])$. On the top, the pre- and post-conditions are equivalent as explained below. In session $\{y[0..n], x[0..n]\}$, range $y[0..n]$ stores the basis state $(\!|a^j \ \%\ N|\!)$, which contains value j that represents the basis states for range $x[0..n]$. Selecting a basis state $a^t \ \%\ N$ also filters the j in range $x[0..n]$, i.e., we collect any j having the relation $a^j \ \%\ N = a^t \ \%\ N$. Notice that modulo multiplication is a periodical function, which means that the relation can be rewritten $a^{t+kp} \ \%\ N = a^t \ \%\ N$, and p is the order. Thus, the $x[0..n]$ state is rewritten as a summation of k: $\frac{1}{\sqrt{s}} \Sigma_{k=0}^{s} |t + kp\rangle$. The probability of selecting $(\!|a^j \ \%\ N|\!)$ is $\frac{s}{2^n}$. In Qafny , we set up additional axioms for these periodical theorems to grant this kind of pre- and post-condition equivalence.

Rules for Diffusion. Quantum diffusion operations ($l \leftarrow$ `dis`) reorient the amplitudes of basis states based on the basis state corresponding to l. They are analogized to an aggregate operation of reshape and mean computation, both appeared in some programming languages, such as Python. The aggregate operation first applies a reshape, where elements are regrouped into a normal form, as the first arrow of Figure 9. More specifically, the diffusion function $\mathcal{D}(n, q)$ (Figure 12) first takes an n'-element `CH` type state $\Sigma^{t=0} z_t\, |c_t\rangle$, where n corresponds to the number of qubits in l. Then, we rearrange the state by extending the element number from n' to $m * n$ with probably adding new elements that originally have zero amplitude (the white elements in Figure 9). Here, let's view a basis c_t as a small-endian (LSB) number $\{\!|c_t|\!\}$. The rearrangement of changing bases c_t (for all t) to $(\!|j|\!).c_{ij}$ is analogized to rewrite a number $\{\!|c_t|\!\}$ to be the form $2^n i + j$, with $j \in [0, 2^n)$, i.e., the reshape step rearranges the basis states to be a periodical counting sequence, with 2^n being the order. **[Yi: This bit-level construction is difficult to understand without context. Hopefully ?? would indeed help.]** The mean computation analogy (the second arrow in Figure 9) takes every period in the reshaped state, and for each basis state in a period, we redistribute its amplitude by the formula $(\frac{1}{2^{n-1}} \sum_{u=0}^{2^n} z_{iu} - z_{ij})$. Rule SDis is the semantics for diffusion $l \leftarrow$ `dis`, which applies the $\mathcal{D}$ function to session $\kappa \uplus \kappa'$, where κ contains

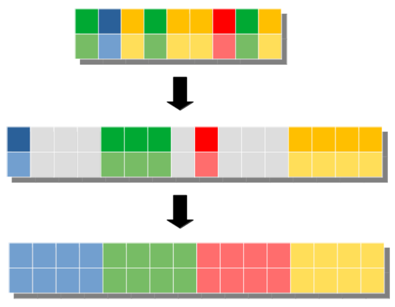
Fig. 12. Diffusion Analogy

all qubits in l. Proof rule PDɪs is a separation style rule that does the same as rule SDɪs. An example of quantum walk algorithm that uses diffusion operations is given in ??.

1.4 Qafny Metatheory

Here, we show the type soundness and proof system soundness and completeness.
Type Soundness. We prove that well-typed Qafny programs are well defined; i.e., the type system is sound with respect to the semantics, with the well-formedness assumptions (?? and Definition 1.1). The Qafny type soundness is stated as two theorems, type progress and preservation . The proofs are done by induction on statements s and mechanized in Coq.

Theorem 1.2 (Qafny type progress). If $\Omega;\sigma \vdash_g s \triangleright \sigma'$ and $\Omega;\sigma \vdash \varphi$, then either $s = \{\}$, or there exists φ' and s' such that $(\varphi, s) \longrightarrow (\varphi', s')$.

Theorem 1.3 (Qafny type preservation). If $\Omega;\sigma \vdash_g s \triangleright \sigma'$, $\Omega;\sigma \vdash \varphi$, and $(\varphi, s) \longrightarrow (\varphi', s')$, then there exists Ω' and σ'', $\Omega';\sigma'' \vdash_g s' \triangleright \sigma'$ and $\Omega';\sigma'' \vdash \varphi'$.

Proof System Soundness and Completeness. We prove that the Qafny proof system is sound and complete with respect to it semantics for well-typed Qafny programs. In Qafny , there are three different state representations for a session κ and two sessions can be joined into a large session. Hence, given a statement s and an initial state ψ and φ, the semantic transition $(\psi, \varphi, s) \longrightarrow^* (\psi', \varphi', \{\})$ might not be unique, such that there might be different φ' representations. However, any `Nor` and `Had` type state can be represented as a `CH` type state, so that `CH` type states can be viewed as the *most general* state representation. We also have state equivalence relations defined for capturing the behaviors of session permutation, join and split. We define the *most general state representation* of evaluating a statement s in an initial state φ below.

Definition 1.4 (Most general Qafny state). Given s, φ, Ω, σ, and g, such that $\Omega;\sigma \vdash_g \varphi$, $\Omega;\sigma \vdash_g s \triangleright \sigma^*$, $\Omega;\sigma[\uparrow \sigma^*] \vdash \varphi^*$, and $(\psi, \varphi, s) \longrightarrow^* (\psi', \varphi^*, \{\})$, φ^* is the most general state representation of evaluating (ψ, φ, s), iff for all σ' and φ', such that $\Omega;\sigma \vdash_g s \triangleright \sigma'$, $\Omega;\sigma[\uparrow \sigma'] \vdash \varphi'$ and $(\psi, \varphi, s) \longrightarrow^* (\psi', \varphi', \{\})$, $\sigma' \preceq \sigma^*$ and $\varphi' \equiv \varphi^*$.

The Qafny proof system correctness is defined by the soundness and relatively completeness theorems below, which has been formalized and proved in Coq. The Qafny proof system only describes the quantum portion built on top of the Dafny system. Since the quantum portion contains non-terminated programs, the soundness and completeness essentially refers to the partial correctness of the Qafny proof system and the total correctness is achieved by compiling Qafny programs to Dafny. The Qafny proof system correctness is defined in terms of programs being well-typed. The type soundness theorem suggests that any intermediate transitions of evaluating a well-typed Qafny program is also well-typed. Thus, the proof system correctness proof only reason about well-typed predicates in Definition 1.1.

Theorem 1.5 (proof system soundness). For a well-typed program s, such that $\Omega;\sigma \vdash_g s \triangleright \sigma'$, $\Omega;\sigma \vdash_g \{P\}\, s\, \{Q\}$, $\Omega;\sigma;\psi;\varphi \models_g P$, then there exists a state representation φ', such that $(\psi, \varphi, s) \longrightarrow^* (\psi', \varphi', \{\})$ and $\Omega;\sigma[\uparrow \sigma'];\psi';\varphi' \models_g Q$, and there is a most general state representation φ^* of evaluating (ψ, φ, s) as $(\psi, \varphi, s) \longrightarrow^* (\psi', \varphi^*, \{\})$ and $\varphi' \equiv \varphi*$.

Theorem 1.6 (proof system relative completeness). For a well-typed program s, such that $\Omega;\sigma \vdash_g s \triangleright \sigma'$, $(\psi, \varphi, s) \longrightarrow^* (\psi', \varphi', \{\})$ and $\Omega;\sigma \vdash_g \varphi$, there is most general state representation φ^*, such that $(\psi, \varphi, s) \longrightarrow^* (\psi', \varphi', \{\})$ and $\varphi' \equiv \varphi^*$ and $\Omega;\sigma \vdash_g s \triangleright \sigma^*$ and $\Omega;\sigma[\uparrow \sigma^*] \vdash_g \varphi^*$, and there are predicates P and Q, such that $\Omega;\sigma;\psi;\varphi \models_g P$ and $\Omega;\sigma[\uparrow \sigma^*];\psi';\varphi^* \models Q$ and $\Omega;\sigma \vdash_g \{P\}\, s\, \{Q\}$.

[Yi: These theorems sound quite interesting and well-motivated from the earlier description, but I'm having difficulty parsing them. What is the ↑ notation?]

REFERENCES

[1] Stephane Beauregard. 2003. Circuit for Shor's Algorithm Using 2n+3 Qubits. *Quantum Info. Comput.* 3, 2 (March 2003), 175–185.

[2] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. 2020. Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 286–300. https://doi.org/10.1145/3385412.3386007

[3] Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. 2021. An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12648)*, Nobuko Yoshida (Ed.). Springer, 148–177. https://doi.org/10.1007/978-3-030-72019-3_6

[4] Andrew Childs, Ben Reichardt, Robert Spalek, and Shengyu Zhang. 2007. Every NAND formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a Quantum Computer. (03 2007).

[5] Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Michał Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. 2009. VCC: A Practical System for Verifying Concurrent C. In *Theorem Proving in Higher Order Logics*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 23–42.

[6] Andrew Cross. 2018. The IBM Q experience and QISKit open-source quantum computing software. In *APS Meeting Abstracts*.

[7] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. 2017. Open quantum assembly language. *arXiv e-prints* (Jul 2017). arXiv:1707.03429 [quant-ph]

[8] Andrew W. Cross, Ali Javadi-Abhari, Thomas Alexander, Niel de Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, John Smolin, Jay M. Gambetta, and Blake R. Johnson. 2021. OpenQASM 3: A broader and deeper quantum assembly language. arXiv:2104.14722 [quant-ph]

[9] J. I. den Hartog. 1999. Verifying Probabilistic Programs Using a Hoare like Logic. In *Advances in Computing Science — ASIAN'99*, P. S. Thiagarajan and Roland Yap (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 113–125.

[10] Thomas G. Draper. 2000. Addition on a Quantum Computer. *arXiv: Quantum Physics* (2000).

[11] Yuan Feng and Mingsheng Ying. 2021. Quantum Hoare Logic with Classical Variables. *ACM Transactions on Quantum Computing* 2, 4, Article 16 (dec 2021), 43 pages. https://doi.org/10.1145/3456877

[12] Craig Gidney and Martin Ekerå. 2021. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* 5 (April 2021), 433. https://doi.org/10.22331/q-2021-04-15-433

[13] Google Quantum AI. 2019. Cirq: An Open Source Framework for Programming Quantum Computers. https://quantumai.google/cirq

[14] Alexander Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2013)*. 333–342. https://doi.org/10.1145/2491956.2462177

[15] Daniel M. Greenberger, Michael A. Horne, and Anton Zeilinger. 1989. *Going beyond Bell's Theorem*. Springer Netherlands, Dordrecht, 69–72. https://doi.org/10.1007/978-94-017-0849-4_10

[16] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866 arXiv:quant-ph/9605043

[17] Lov K. Grover. 1997. Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Phys. Rev. Lett.* 79 (July 1997), 325–328. Issue 2. https://doi.org/10.1103/PhysRevLett.79.325 arXiv:quant-ph/9706033

[18] Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. 2021. Proving Quantum Programs Correct. In *Proceedings of the Conference on Interative Theorem Proving (ITP)*.

[19] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021. A Verified Optimizer for Quantum Circuits. In *Proceedings of the ACM Conference on Principles of Programming Languages (POPL)*.

[20] C. A. R. Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580. https://doi.org/10.1145/363235.363259

[21] Shachar Itzhaky, Hila Peleg, Nadia Polikarpova, Reuben N. S. Rowe, and Ilya Sergey. 2021. Cyclic Program Synthesis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 944–959. https://doi.org/10.1145/3453483.3454087

[22] Yoshihiko Kakutani. 2009. A Logic for Formal Verification of Quantum Programs. In *Advances in Computer Science - ASIAN 2009. Information Security and Privacy*, Anupam Datta (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 79–93.

[23] Emmanuel Knill. 1996. *Conventions for quantum pseudocode.* Technical Report. Los Alamos National Lab., NM (United States).

[24] Xuan-Bach Le, Shang-Wei Lin, Jun Sun, and David Sanan. 2022. A Quantum Interpretation of Separating Conjunction for Local Reasoning of Quantum Programs Based on Separation Logic. *Proc. ACM Program. Lang.* 6, POPL, Article 36 (jan 2022), 27 pages. https://doi.org/10.1145/3498697

[25] Xuan-Bach Le, Shang-Wei Lin, Jun Sun, and David Sanan. 2022. A Quantum Interpretation of Separating Conjunction for Local Reasoning of Quantum Programs Based on Separation Logic. *Proc. ACM Program. Lang.* 6, POPL, Article 36 (jan 2022), 27 pages. https://doi.org/10.1145/3498697

[26] K. Rustan M. Leino. 2010. Dafny: An Automatic Program Verifier for Functional Correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Edmund M. Clarke and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 348–370.

[27] K. Rustan M. Leino and Michał Moskal. 2014. Co-induction Simply. In *FM 2014: Formal Methods*, Cliff Jones, Pekka Pihlajasaari, and Jun Sun (Eds.). Springer International Publishing, Cham, 382–398.

[28] Liyi Li, Finn Voichick, Kesha Hietala, Yuxiang Peng, Xiaodi Wu, and Michael Hicks. 2022. Verified Compilation of Quantum Oracles. In *OOPSLA 2022.* https://doi.org/10.48550/ARXIV.2112.06700

[29] Yangjia Li and Dominique Unruh. 2021. Quantum Relational Hoare Logic with Expectations. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 198)*, Nikhil Bansal, Emanuela Merelli, and James Worrell (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 136:1–136:20. https://doi.org/10.4230/LIPIcs.ICALP.2021.136

[30] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. 2019. Formal Verification of Quantum Algorithms Using Quantum Hoare Logic. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 187–207.

[31] Christof Löding, P. Madhusudan, and Lucas Peña. 2017. Foundations for Natural Proofs and Quantifier Instantiation. *Proc. ACM Program. Lang.* 2, POPL, Article 10 (dec 2017), 30 pages. https://doi.org/10.1145/3158098

[32] Parthasarathy Madhusudan, Xiaokang Qiu, and Andrei Stefanescu. 2012. Recursive Proofs for Inductive Tree Data-Structures. *SIGPLAN Not.* 47, 1 (jan 2012), 123–136. https://doi.org/10.1145/2103621.2103673

[33] Igor L. Markov and Mehdi Saeedi. 2012. Constant-Optimized Quantum Circuits for Modular Multiplication and Exponentiation. *Quantum Info. Comput.* 12, 5–6 (May 2012), 361–394.

[34] Narciso Martí-Oliet and José Meseguer. 2000. Rewriting logic as a logical and semantic framework. In *Electronic Notes in Theoretical Computer Science*, J. Meseguer (Ed.), Vol. 4. Elsevier Science Publishers.

[35] Microsoft. 2017. *The Q# Programming Language.* https://docs.microsoft.com/

[36] Daniel Neider, Pranav Garg, P. Madhusudan, Shambwaditya Saha, and Daejun Park. 2018. Invariant Synthesis for Incomplete Verification Engines. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dirk Beyer and Marieke Huisman (Eds.). Springer International Publishing, Cham, 232–250.

[37] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information* (10th anniversary ed.). Cambridge University Press, USA.

[38] Edgar Pek, Xiaokang Qiu, and P. Madhusudan. 2014. Natural Proofs for Data Structure Manipulation in C Using Separation Logic. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh, United Kingdom) *(PLDI '14).* Association for Computing Machinery, New York, NY, USA, 440–451. https://doi.org/10.1145/2594291.2594325

[39] Yuxiang Peng, Kesha Hietala, Runzhou Tao, Liyi Li, Robert Rand, Michael Hicks, and Xiaodi Wu. 2022. A Formally Certified End-to-End Implementation of Shor's Factorization Algorithm. https://doi.org/10.48550/ARXIV.2204.07112

[40] Xiaokang Qiu, Pranav Garg, Andrei Ştefănescu, and Parthasarathy Madhusudan. 2013. Natural Proofs for Structure, Data, and Separation. *SIGPLAN Not.* 48, 6 (jun 2013), 231–242. https://doi.org/10.1145/2499370.2462169

[41] Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and Parthasarathy Madhusudan. 2013. Natural proofs for structure, data, and separation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, Hans-Juergen Boehm and Cormac Flanagan (Eds.). ACM, 231–242. https://doi.org/10.1145/2491956.2462169

[42] Robert Rand. 2018. *Formally verified quantum programming.* Ph.D. Dissertation. University of Pennsylvania.

[43] Rayleigh. [n.d.]. The Problem of the Random Walk. *Nature* 72 ([n. d.]), 318–318.

[44] J.C. Reynolds. 2002. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science.* 55–74. https://doi.org/10.1109/LICS.2002.1029817

[45] Rigetti Computing. 2021. PyQuil: Quantum programming in Python. https://pyquil-docs.rigetti.com

[46] Grigore Roşu and Andrei Ştefănescu. 2011. Matching Logic: A New Program Verification Approach (NIER Track). In *ICSE'11: Proceedings of the 30th International Conference on Software Engineering.* ACM, 868–871. https://doi.org/doi:10.1145/1985793.1985928

[47] Grigore Roşu, Andrei Ştefănescu, Ştefan Ciobâcă, and Brandon M. Moore. 2013. One-Path Reachability Logic. In *Proceedings of the 28th Symposium on Logic in Computer Science (LICS'13)*. IEEE, 358–367.

[48] P.W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 124–134. https://doi.org/10.1109/SFCS.1994.365700

[49] P. W. Shor. 1994. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science (FOCS '94)*.

[50] Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. 2016. Automated Mutual Explicit Induction Proof in Separation Logic. https://doi.org/10.48550/ARXIV.1609.00919

[51] Dominique Unruh. 2019. Quantum Relational Hoare Logic. *Proc. ACM Program. Lang.* 3, POPL, Article 33 (jan 2019), 31 pages. https://doi.org/10.1145/3290346

[52] Salvador Elías Venegas-Andraca. 2012. Quantum walks: a comprehensive review. *Quantum Information Processing* 11, 5 (jul 2012), 1015–1106. https://doi.org/10.1007/s11128-012-0432-5

[53] Thomas G. Wong. 2022. Unstructured search by random and quantum walk. *Quantum Information and Computation* 22, 1&2 (jan 2022), 53–85. https://doi.org/10.26421/qic22.1-2-4

[54] Mingsheng Ying. 2012. Floyd–Hoare Logic for Quantum Programs. *ACM Trans. Program. Lang. Syst.* 33, 6, Article 19 (Jan. 2012), 49 pages. https://doi.org/10.1145/2049706.2049708

[55] Mingsheng Ying. 2019. Toward Automatic Verification of Quantum Programs. *Form. Asp. Comput.* 31, 1 (feb 2019), 3–25. https://doi.org/10.1007/s00165-018-0465-3

[56] Bohua Zhan. 2018. Efficient Verification of Imperative Programs Using Auto2. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dirk Beyer and Marieke Huisman (Eds.). Springer International Publishing, Cham, 23–40.

[57] Li Zhou, Gilles Barthe, Justin Hsu, Mingsheng Ying, and Nengkun Yu. 2021. A Quantum Interpretation of Bunched Logic and Quantum Separation Logic. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science* (Rome, Italy) *(LICS '21)*. Association for Computing Machinery, New York, NY, USA, Article 75, 14 pages. https://doi.org/10.1109/LICS52264.2021.9470673