Quantum Natural Proof

ANONYMOUS AUTHOR(S)

Q-Dafny.

1 QWHILE: A HIGH-LEVEL QUANTUM LANGUAGE

We introduce the language syntax and type system for QWhile and introduce the Q-Dafny Proof system. As a running example, we specify Shor's algorithm and its proof in Q-Dafny in Figure 2. The Q-Dafny to Dafny compiler is under construction, but the compiled version of the Shor's algorithm proof has been finalized and can be found at https://github.com/inQWIRE/VQO/blob/naturalproof/Q-Dafny/examples/Shor-compiled.dfy.

1.1 Sessions, Kinds, Types, and States

The Qafny element component syntax is represented according to the grammar in Figure 3. In Qafny , there are three kinds of values, two of which are classical ones represented by the two modes: c and q. The former represents classical values, represented as a natural number n, that do not intervene with quantum measurements and are evaluated in the compilation time, the latter represents values, represented as a pair (r, n), produced from a quantum measurement. The real number r is a characteristic representing the theoretical probability of the measurement resulting in the value n. Any classical arithmetic operation does not change r, i.e., $(r, n) + m = (r, n + m)$.

Quantum values are defined in terms of sessions (λ), which can be viewed as clusters of possibly entangled qubits, where the number of qubits is exactly the session length, i.e., $\texttt{len}(\overline{x[n..m]})$. Each session consists of different disjoint ranges represented as $x[n..m]$ that refers the range $[n, m)$ in a quantum array named x. For simplicity, we assume that different variable names referring to different quantum arrays without aliasing. Each length-n session is associated to a quantum state that can be one of the three forms (q in Figure 3) that are corresponding to three different types (τ in Figure 3). The first kind of state is of Nor type (Nor (c opt)), having the state form $|c\rangle$, which is a computational basis value. c is of length n and represents a tensor product of qubits, all being 0 or 1. The second kind of state is of Had type (Had (◯ opt)), meaning that qubits in such session are in superposition but not entangled. The state form is $\frac{1}{\sqrt{2^n}} \bigotimes_{j=0}^{n} (|0\rangle + \alpha(r_j)\,|1\rangle)$, where $\alpha(r_j)$ is a local phase for the j-th qubit in the session. If $r_j = 0$ for all j, the state can be represented by type Had ◯ representing a uniformly distributed superposition; otherwise, we represent the type as Had ∞. The third kind of state is of CH type (CH ($\overline{c}(m)$ opt)), having the state form $\sum_{j=0}^{m} z_j\,|c_j\rangle$, referring to that qubits in such session are possibly entangled. The state $\sum_{j=0}^{m} z_j\,|c_j\rangle$ can be viewed as an m element set of pairs $z_j\,|c_j\rangle$, where z_j and c_j are the j-th amplitude and basis. The well-formed restrictions for the state are three: 1) $\sum_{j=0}^{m} |z_j|^2 = 1$ (z_j is a complex number); 2) length of c_j is n for all j and $m \leq 2^n$; 3) any two bases c_j and c_k are distinct.

In Qafny , the quantum types and states are associated through bases and equational properties. For each quantum state q, especially for Nor type state $|c\rangle$ and CH type state $\sum_{j=0}^{m} z_j\,|c_j\rangle$, the type factors are either ∞ meaning no bases can be tracked, or having the form c and $\overline{c}(m)$ that track the bases of the state $|c\rangle$ and $\sum_{j=0}^{m} z_j\,|c_j\rangle$, respectively. For Nor type, this means that the type factor c (in Nor c) and the state qubit format $|c\rangle$ must be equal; for CH type (CH $\overline{c}(m)$), if the state is $\sum_{j=0}^{m} z_j\,|c_j\rangle$, the j-th element $\overline{c}[j]$ is equal to c_j. Additionally, Qafny types permit subtyping relations that correspond to state equivalent relations in Figure 4. Both subtype relation $\sqsubseteq_n$ and state equivalence relation $\equiv_n$ are parameterized by a session length number n, such that they establish relations between two quantum states describing a session of length n. $\sqsubseteq_n$ in Figure 4a describes a type term

Bitstring	c	$\in$	$(0 \mid 1)^+$
Indexed bitstring set	$\overline{c}(m)$	$::=$	$\{c_0, c_1, ..., c_{m-1}\}$
Nat. Num	m, n	$\in$	$\mathbb{N}$
Real	r	$\in$	$\mathbb{R}$
Complex Number	z	$\in$	$\mathbb{C}$
Phase	$\alpha(r)$	$::=$	$e^{2\pi i r}$
Program/Session Variable	x, y		
Mode	g	$::=$	$\mathtt{c} \mid \mathtt{q}$
Classical Value	v	$::=$	$n \mid (r, n)$
Session	λ	$::=$	$\overline{x[n..m]}$
Full Mode (Kind)	k	$::=$	$g \mid \lambda$
Option	$p \in {}'a\ \mathtt{opt}$	$::=$	$'a \mid \infty$
Uniform Distribution	$\bigcirc$		
Type	τ	$::=$	$\mathtt{Nor}\ (c\ \mathtt{opt}) \mid \mathtt{Had}\ (\bigcirc\ \mathtt{opt}) \mid \mathtt{CH}\ (\overline{c}(m)\ \mathtt{opt})$
Quantum States	q	$::=$	$\vert c\rangle \mid \frac{1}{\sqrt{2^n}} \bigotimes_{j=0}^{n} (\vert 0\rangle + \alpha(r_j)\vert 1\rangle) \mid \sum_{j=0}^{m} z_j \vert c_j\rangle$

Fig. 1. QAFNY element syntax. In $\overline{c}(m)$, $\overline{c}$ is a bitstring set and m is the element number, and it can be abbreviated as $\overline{c}$. Each element $x[n..m]$ in a session $\overline{x[n..m]}$ represents the range $[n, m)$ in a qubit array x.

$$
\begin{array}{lll}
\mathtt{Nor}\ \infty & \sqsubseteq_n & \mathtt{CH}\ \infty \\
\mathtt{Nor}\ c & \sqsubseteq_n & \mathtt{CH}\ \{c\} \\
\mathtt{CH}\ \overline{c}(1) & \sqsubseteq_n & \mathtt{Nor}\ \overline{c}[0] \\
\mathtt{Had}\ p & \sqsubseteq_n & \mathtt{CH}\ \{c(i) | i \in [0, 2^n)\}(2^n) \\
\mathtt{CH}\ \{0, 1\} & \sqsubseteq_1 & \mathtt{Had}\ \infty \\
\mathtt{CH}\ p & \sqsubseteq_n & \mathtt{CH}\ \infty
\end{array}
$$

(a) Subtyping

$$
\begin{array}{lll}
|c\rangle & \equiv_n & \sum_{j=0}^{1} |c\rangle \\
\sum_{j=0}^{1} z_j |c_j\rangle & \equiv_n & |c_0\rangle \\
\frac{1}{\sqrt{2^n}} \bigotimes_{j=0}^{n} (|0\rangle + \alpha(r_j)|1\rangle) & \equiv_n & \sum_{j=0}^{2^n} \frac{\alpha(\sum_{k=0}^{n} r_k \cdot (\!| j |\!)[k])}{\sqrt{2^n}} |j\rangle \\
\sum_{j=0}^{2} z_j |c_j\rangle & \equiv_1 & \frac{1}{\sqrt{2}} \bigotimes_{j=0}^{1} (|0\rangle + \frac{\sqrt{2} z_1}{z_0} |1\rangle) \\
 & & \text{when } c_0 = 0 \ \ c_1 = 1
\end{array}
$$

(b) State Equivalence

Fig. 2. QAFNY type/state relations. $\overline{c}[n]$ produces the n-th element in set $\overline{c}$. $\{c(i) | i \in [0, 2^n)\}(2^n)$ defines a set $\{c(i) | i \in [0, 2^n)\}$ with the emphasis that it has 2^n elements. $\{0, 1\}$ is a set of two single element bitstrings 0 and 1. $\cdot$ is the multiplication operation, $(\!| j |\!)$ turns a number j to a bitstring, $(\!| j |\!)[k]$ takes the k-th element in the bitstring $(\!| j |\!)$, and $|j\rangle$ is an abbreviation of $|(\!| j |\!)\rangle$.

on the left can be used as a type on the right. For example, a Nor type qubit array Nor c can be used as a single element entanglement type term CH $\{c\}$ [1]. Correspondingly, state equivalence relation $\equiv_n$ describes the two state forms to be equivalent; specifically, the left state term can be used as the right one, e.g., a single element entanglement state $\sum_{j=0}^{1} z_j |c_j\rangle$ can be used as a Nor type state $|c_0\rangle$ with the fact that z_0 is now a global phase that can be neglected.

1.2 Syntax

QWhile is a high-level language for describing quantum programs, which permits quantum control and for-loop statements. Figure 5 introduces the QWhile syntax.

A QWhile program consists of a sequence of C-like statements s. Values and variables (ranged by x and y) in a statement are classified as three different categories: compilation time classical values, classical values generated from quantum measurement, and quantum state registers. We use variable *modes* to classify the first two kinds as cand qmodes. We represent c-mode values as

[1] If a qubit array only consists of 0 and 1, it can be viewed as an entanglement of unique possibility.

Nat. Num	m, n	$\in$	$\mathbb{N}$
Real	r	$\in$	$\mathbb{R}$
Variable	x, y		
c-Mode Value	n		
q-Mode Value	(r, n)		
$\mathbb{O}$QASM Expr	μ		
Session	ζ	$::=$	$\overline{(x, n, m)}$
Type Predicate	T	$::=$	$x : \tau \mid x : \{\zeta : \tau\} \mid ...$
State Predicate	P, Q	$::=$	$x = n \mid x = (r, n) \mid \zeta = \rho \mid ...$
Mode	g	$::=$	$\mathtt{c} \mid \mathtt{q}$
Mode Check Result	q	$::=$	$g \mid \zeta$
Factor	l	$::=$	$x \mid x[a]$
Arith Expr	p, a	$::=$	$l \mid a + a \mid a * a \mid ...$
Bool Expr	b	$::=$	$x[a] \mid (a = a)@x[a] \mid (a < a)@x[a] \mid \neg b...$
Gate Expr	op	$::=$	$\mathtt{H} \mid \mathtt{QFT}^{[-1]}$
C/M Moded Expr	e	$::=$	$a \mid \mathtt{measure}(y)$
Statement	s	$::=$	$\{\} \mid \mathtt{let}\ x\ =\ e\ \mathtt{in}\ s \mid l \leftarrow op \mid \bar{l} \leftarrow a(\mu)$
		$\mid$	$s\ ;\ s \mid \mathtt{if}\ (b)\ \{s\} \mid \mathtt{for}\ (\mathtt{int}\ i\ =\ a_1\ ;\ i < a_2\ ;\ b(i)\ ;\ f(i))\ T(i)\ P(i)\ \{s\}$
		$\mid$	$\mathtt{amplify}\{x \leftarrow a\} \mid \mathtt{diffuse}(l)$

ρ: Quantum States see Figure 13

Fig. 3. Core QWhile Syntax, $\mathbb{O}$QASM Syntax is in Figure 14

Bit	d	$::=$	$0 \mid 1$
BitString	$\overline{d}$	$::=$	$\mathbb{N} \to d$
BitString Indexed Set	β	$::=$	$\{\overline{d}\} \mid \infty$
Phase Type	w	$::=$	$\bigcirc \mid \infty$
Type Element	t	$::=$	$\mathtt{Nor}\ \overline{d} \mid \mathtt{Had}\ w \mid \mathtt{CH}\ n\beta$
Type	τ	$::=$	$\bigotimes_n\ t$
Phase	$\alpha(n)$	$:=$	$e^{2\pi i \frac{1}{n}}$
Amplitude	θ	$:=$	r
Phi State	$\|\Phi(n)\rangle$	$:=$	$\frac{1}{\sqrt{2}}(\|0\rangle + \alpha(n)\|1\rangle)$
Quantum State	ρ	$::=$	$\alpha \|\overline{d}\rangle \mid \bigotimes_{k=0}^{m} \|\Phi(n_k)\rangle \mid \sum_{k=0}^{m} \theta_k \|\overline{d_k}\rangle$

Fig. 4. QWhile Sessions and Types and Quantum States

natural numbers, while M-mode values are represented as pairs of reals and natural numbers. The reals represent the conceptual occurrence probability of the result measurement, and the natural numbers are the measurement results. Any further arithmetic operations on M-mode values are applied on the measurement results, such as $(r, n_1) + n_2 = (r, n_1 + n_2)$.

Quantum registers refer to quantum qubit arrays in QWhile. They are always associated with disjoint sets of qubit array fragments, named *sessions*. We represent a qubit array fragment as (x, n, m), where x is a variable representing a qubit array, n is the start position in x for the array fragment and m is the exclusive ending point. In Figure 2, an example array fragment for `x` is `x[0..i]`. `y` is an abbreviation of array fragment `y[0..n]` where `n` is the ending point of array `y`. `(y,x[0..i])` is a session containing two array fragments. It is worth noting that the ordering of fragments in a session only serves for recording the qubit positions in a quantum register. The

$$\frac{}{\Omega \vdash x : \Omega(x)} \qquad \frac{\Omega(x) = (x, 0, \Sigma(x))}{\Omega \vdash x[n] : [(x, n, n+1)]} \qquad \frac{\Omega \vdash a_1 : q_1 \qquad \Omega \vdash a_2 : q_2}{\Omega \vdash a_1 + a_2 : q_1 \sqcup q_2} \qquad \frac{\Omega \vdash a_1 : q_1 \qquad \Omega \vdash a_2 : q_2}{\Omega \vdash a_1 * a_2 : q_1 \sqcup q_2}$$

$$\frac{\Omega \vdash a_1 : q_1 \qquad \Omega \vdash a_2 : q_2 \Omega \vdash a_3 : q_3}{\Omega \vdash (a_1 = a_2)@x[n] : q_1 \sqcup q_2 \sqcup q_3} \qquad \frac{\Omega \vdash a_1 : q_1 \qquad \Omega \vdash a_2 : q_2 \Omega \vdash a_3 : q_3}{\Omega \vdash (a_1 < a_2)@x[n] : q_1 \sqcup q_2 \sqcup q_3} \qquad \frac{\Omega \vdash b : q}{\Omega \vdash \neg b : q} \qquad \frac{\Omega \vdash e : \zeta_2 \sqcup \zeta_1}{\Omega \vdash e : \zeta_1 \sqcup \zeta_2}$$

$$\zeta_1 \sqcup \zeta_2 = \zeta_1 \uplus \zeta_2 \zeta \sqcup g = \zeta \qquad g \sqcup \zeta = \zeta \qquad \mathsf{c} \sqcup \mathsf{c} = \mathsf{c} \qquad \mathsf{q} \sqcup \mathsf{c} = \mathsf{q} \qquad \mathsf{c} \sqcup \mathsf{q} = \mathsf{q} \qquad \mathsf{c} \leq \mathsf{q} \leq \zeta$$
$$\bot \uplus l = l \qquad l \uplus \bot = l \qquad [(x, v_1, v_2)] \uplus [(y, v_3, v_4)] = [(x, v_1, v_2), (y, v_3, v_4)]$$
$$[v_2, v_2] \cap [v_3, v_4] \neq \emptyset \Rightarrow [(x, v_1, v_2)] \uplus [(x, v_3, v_4)] = [(x, \mathtt{min}(v_1, v_3), \mathtt{max}(v_2, v_4))]$$

Fig. 5. Arith, Bool, Gate Mode Checking

swapping of fragment ordering does not affect the register state. In Q-Dafny, the fragment ordering might affect the property proof automation. In Figure 2, we fix fragment order as `(y,x[0..i])` to enhance the Shor's algorithm proof automation.

We use $x[a]$ to represent the a-th position qubit in x. It is worth noting that the variable x in $x[a]$ must represent quantum registers. We name a variable x or an array index $x[a]$ as a factor. In a QWhile program, $\mathsf{c}$and $\mathsf{q}$mode variables act like stack variables and they must be bounded by variables introduced by `let` statements; while quantum registers represent arrays in a "quantum heap" and are bounded by Σ.

The statements s in the first row in Figure 5 are `{}`(SKIP) and assignment operations. Classic assignment (`let x = e in s`) evaluates e and assigns the value to $\mathsf{c}$or $\mathsf{q}$mode variable x that is used in s. Expressions e can be an arithmetic or a measurement operation. `let` x `= measure(`y`) in` ... assigns the measurement result of qubit array y to x. $l \leftarrow op$ and $\bar{l} \leftarrow a(\mu)$ are quantum assignments. The former applies a simple quantum gate (`H` or `QFT`) to a single qubit ($x[a]$) or a qubit array (x) [2]. $\bar{l} \leftarrow a(\mu)$ is a generalized quantum assignment that implements quantum oracle circuits a and applies an $\mathcal{O}$QASM/$\mathcal{O}$QIMP operation on a list of qubit array $\bar{l}$. We assume all arithmetic (a) and Boolean (b) expressions are reversible. For example, the operation $(a_1 < a_2)@x[a]$ compares a_1 and a_2 and stores the value in $x[a]$. μ is the circuit implementation of the expression a in $\mathcal{O}$QASM, whose syntax is given in Section 3. One can utilize $\mathcal{O}$QASM expressions (μ) to implement singleton `X` and $\mathtt{RZ}^{[-1]}\ n$ gates, thus, the QWhile syntax is universal with respect to quantum gates. In addition, $\mathcal{O}$QIMP is a C-like reversible arithmetic language built on $\mathcal{O}$QASM. The reversible expression a in Figure 5 is based on $\mathcal{O}$QIMP operations. For simplicity, we write $\bar{l} \leftarrow a$ in examples here to mean that it applies a $\mathcal{O}$QIMP circuit that computes a to $\bar{l}$.

The second row of statements in Figure 5 are control-flow operations. s_1 ; s_2 is a sequential operation. `if` (b) $\{s\}$ is a conditional and b might contain quantum factors. Every quantum factor l appearing in b must not appear in s. In QWhile, we define a `well_formed` predicate to check such property. `for (int` i $=$ a_1 ; $i < a_2$; $b(i)$; $f(i)$) $T(i)$ $P(i)$ $\{s\}$ is a possibly quantum for-loop depending on if the Boolean guard $b(i)$ ($b(i)$ is a Boolean expression that contains variable i) contains quantum factors (also needing to be `well_formed`). A classical variable i is introduced and it is initialized as the lower bound a_1, increments in each loop step by the monotonic increment function $f(i)$, and ends at the upper bound a_2. $T(i)$ is the type predicate (type predicate syntax in fig. 5) where for i-th loop step, $T(i)$ is true in the beginning and $T(f(i))$ is maintained after the execution. Similarly, $P(i)$ is the loop-invariant (state predicate in fig. 5) for the for-loop structure, where for i-th loop step, $P(i)$ is true in the beginning and $P(f(i))$ is maintained after the execution.

[2] `QFT` gate must apply to a variable x

The last row contains quantum reflection operations, which are used to adjust the occurrence probability of some quantum states in a quantum qubit array. For example, in Grover's search algorithm [?], the Grover's diffusion operation is a quantum reflection that increases the occurrence probability of a target quantum state in a qubit array being in superposition. We identify two kinds of quantum reflections that previous works tent to combine together. The first one is an amplifier (`amplify`$\{x \leftarrow a\}$) that amplifies the occurrence probability of the target state, which is represented by a classical value a, in a quantum qubit array x. The second one is a diffusion operator (`diffuse`(x)) that diffuses the state of a qubit array x to a superposition of all possible bases in x with possibly different amplitudes [3]. For example, applying the diffusion operator on a two-qubit array x having state $|00\rangle$ results in a superposition of $-\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$. In general, if an n-qubit array x has the state $\sum_{j=0} \alpha_j |x_j\rangle$, the result of applying a diffusion operator is $\frac{1}{2^n}(2\sum_{i=0}^{2^n}(\sum_{j=0}\alpha_j)|i\rangle - \sum_{j=0}\alpha_j|x_j\rangle)$.

1.3 Type Checking: A Quantum Session Type System

The QWhile type system can be viewed as a mapping from lists of factors (x or $x[n]$) to QWhile types in Figure 13. Generally, factors represent a range of locations in a "quantum heap". Variable x can be viewed as a range $(x, 0, \Sigma(x))$, meaning the heap range starting at x and ending at $x + n$. Index $x[n]$ can be viewed as a range $(x, n, n + 1)$. Thus, a list of **quantum** factors is essentially a list of disjoint fragments, which it is called a *session*.

A type is written as $\bigotimes_n\ t$, where n refers to the total number of qubits in a session, and t describes the qubit state form. A session being type $\bigotimes_n$ `Nor` $\overline{d}$ means that every qubit is in normal basis (either $|0\rangle$ or $|1\rangle$), and $\overline{d}$ describes basis states for the qubits. The type corresponds to a single qubit basis state $\alpha(n)\,|\overline{d}\rangle$, where the global phase $\alpha(n)$ has the form $e^{2\pi i \frac{1}{n}}$ and $\overline{d}$ is a list of bit values. Global phases for `Nor` type are usually ignored in many semantic definitions. In QWhile, we record it because in quantum conditionals, such global phases might be turned to local phases.

$\bigotimes_n$ `Had` w means that every qubit in the session has the state: $(\alpha_1|0\rangle + \alpha_2|1\rangle)$; the qubits are in superposition but they are not entangled. $\bigcirc$ represents the state is a uniform superposition, while ∞ means the phase amplitude for each qubit is unknown. If a session has such type, it then has the value form $\bigotimes_{k=0}^{m} |\Phi(n_k)\rangle$, where $|\Phi(n_k)\rangle = \frac{1}{\sqrt{2}}(|0\rangle + \alpha(n_k)|1\rangle)$.

All qubits in a session that has type $\bigotimes_n$ `CH` $m\beta$ are supposedly entangled (eventual entanglement below). m refers to the number of possible different entangled states in the session, and the bitstring indexed set β describes each of these states, while every element in β is indexed by $i \in [0, m)$. β can also be ∞ meaning that the entanglement structure is unknown. For example, in quantum phase estimation, after applying the $\texttt{QFT}^{-1}$ operation, the state has type $\bigotimes_n$ `CH` $m\infty$. In such case, the only quantum operation to apply is a measurement. If a session has type $\bigotimes_n$ `CH` $m\beta$ and the entanglement is a uniform superposition, we can describe its state as $\sum_{i=0}^{m}\frac{1}{\sqrt{m}}\beta(i)$, and the length of bitstring $\beta(i)$ is n. For example, in a n-length GHZ application, the final state is: $|0\rangle^{\otimes n} + |1\rangle^{\otimes n}$. Thus, its type is $\bigotimes_n$ `CH` $2\{\overline{0}^n, \overline{1}^n\}$, where $\overline{d}^n$ is a n-bit string having bit d.

The type $\bigotimes_n$ `CH` $m\beta$ corresponds to the value form $\sum_{k=0}^{m}\theta_k\,|\overline{d_k}\rangle$. θ_k is an amplitude real number, and $\overline{d_k}$ is the basis. Basically, $\sum_{k=0}^{m}\theta_k\,|\overline{d_k}\rangle$ represents a size m array of basis states that are pairs of θ_k and $\overline{d_k}$. For a session ζ of type `CH`, one can use $\zeta[i]$ to access the i-th basis state in the above summation, and the length is m. In the Q-Dafny implementation section, we show how we can represent θ_k for effective automatic theorem proving.

[3]The possible bases do not depend on x's state, and it is only related to the qubit size of x; i.e., if x is a two qubit array, then the operation reflects the superposition of all possible two qubit states as: $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$.

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9
$x[i]$ y	Nor any	Had Nor	Had Nor	Had Had	Had Had	Had CH	Had CH	CH CH	CH CH
y's operation type	any	$\mathcal{X}$	$\mathcal{R}$	$\mathcal{X}$	$\mathcal{R}$	$\mathcal{X}$	$\mathcal{R}$	$\mathcal{X}$	$\mathcal{R}$
Output Type Entangled?	N	Y	N	N	Y	Y	Y	Y	Y

Fig. 6. Control Gate Entanglement Situation

The QWhile type system has the type judgment: $\Omega, \mathcal{T} \vdash_g s : \zeta \triangleright \tau$, where g is the context mode, mode environment Ω maps variables to modes or sessions (q in Figure 5), type environment $\mathcal{T}$ maps a session to its type, s is the statement being typed, ζ is the session of s, and τ is ζ's type. The QWhile type system in Figure 10 has several tasks. First, it enforces context mode restrictions. Context mode g is either cor q. Q represents the current expression lives inside a quantum conditional or loop, while crefers to other cases. In a Q context, one cannot perform M-mode operations, i.e., no measurement is allowed. There are other well-formedness enforcement. For example, the session of the Boolean guard b in a conditional/loop is disjoint with the session in the conditional/loop body, i.e., qubits used in a Boolean guard cannot appear in its conditional/loop body.

Second, the type system enforces mode checking for variables and expressions in Figure 16. In QWhile, c-mode variables are evaluated to values during type checking. In a `let` statement (Figure 10), c-mode expression is evaluated to a value n, and the variable x is replaced by n in s. The expression mode checking (Figure 16) has the judgment: $\Omega \vdash (a \mid b) : q$. It takes a mode environment Ω, and an expression (a, b), and judges if the expression has the mode g if it contains only classical values, or a quantum session ζ if it contains some quantum values. All the supposedly c-mode locations in an expression are assumed to be evaluated to values in the type checking step, such as the index value $x[n]$ in difference expressions in Figure 16. It is worth noting that the session computation ($\uplus$) is also commutative as the last rule in Figure 16.

Third, by generating the session of an expression, the QWhile type system assigns a type τ for the session indicating its state format, which will be discussed shortly below. Recall that a session is a list of quantum qubit fragments. In quantum computation, qubits can entangled with each other. We utilize type τ (Figure 13) to state entanglement properties appearing in a group of qubits. It is worth noting that the entanglement property refers to *eventual entanglement*, .i.e. a group of qubits that are eventually entangled. Entanglement classification is tough and might not be necessary. In most near term quantum algorithms, such as Shor's algorithm [?] and Childs' Boolean equation algorithm (BEA) [?], programmers care about if qubits eventually become entangled during a quantum loop execution. This is why the normal basis type ($\bigotimes_n$ Nor $\overline{d}$) can also be a subtype of a entanglement type ($\bigotimes_n$ CH $1\{\overline{d}\}$) in our system (Figure 9).

Entanglement Types. We first investigate the relationship between the types and entanglement states. It is well-known that every single quantum gate application does not create entanglement (X, H, and RZ). It is enough to classify entanglement effects through a control gate application, i.e., `if` $(x[i])$ $\{e(y)\}$, where the control node is $x[i]$ and e is an operation applying on y.

A qubit can be described as $\alpha_1 |b_1\rangle + \alpha_2 |b_2\rangle$, where α_1/α_2 are phase amplitudes, and b_1/b_2 are bases. For simplicity, we assume that when we applying a quantum operation on a qubit array y, we either solely change the qubit amplitudes or bases. We identify the former one as $\mathcal{R}$ kind, referring to its similarity of applying an RZ gate; and the latter as $\mathcal{X}$ kind, referring to its similarity of applying an X gate. The entanglement situation between $x[i]$ and y after applying a control statement `if` $(x[i])$ $\{e(y)\}$ is described in Figure 8.

$$\bigotimes_n \ \mathtt{Nor}\ \overline{d} \sqsubseteq \bigotimes_n \ \mathtt{CH}\ 1\{\overline{d}\} \qquad \bigotimes_n \ \mathtt{CH}\ 2^n\beta \sqsubseteq \bigotimes_n \ \mathtt{CH}\ 2^n\infty \qquad \bigotimes_n \ \mathtt{Had}\ \bigcirc \sqsubseteq \bigotimes_n \ \mathtt{CH}\ 2^n\mathscr{P}(n)$$

Fig. 7. Session Type Subtyping

If $x[i]$ has input type Nor, the control operation acts as a classical conditional, i.e., no entanglement is possible. In most quantum algorithms, $x[i]$ will be in superposition (type Had) to enable entanglement creation. When y has type Nor, if y's operation is of $\mathcal{X}$ kind, an entanglement between $x[i]$ and y is created, such as the GHZ algorithm; if the operation is of $\mathcal{R}$ kind, there is not entanglement after the control application, such as the Quantum Phase Estimation (QPE) algorithm.

When $x[i]$ and y are both of type Had, if we apply an $\mathcal{X}$ kind operation on y, it does not create entanglement. An example application is the phase kickback pattern. If we apply a $\mathcal{R}$ operation on y, this does create entanglement. This kind of operations appears in state preparations, such as preparing a register x to have state $\sum_{t=0}^{N} i^{-t}\,|t\rangle$ in Childs' Boolean equation algorithm [?]. The main goal for preparing such state is not to entanglement qubits, but to prepare a state with phases related to its bases.

The case when $x[i]$ and y has type Had and CH, respectively, happens in the middle of executing a quantum loop, such as in the Shor's algorithm and BEA. Applying both $\mathcal{X}$ and $\mathcal{R}$ kind operations result in entanglement. In this narrative, algorithm designers intend to merge an additional qubit $x[i]$ into an existing entanglement session y. $x[i]$ is commonly in uniform superposition, but there can be some additional local phases attached with some bases, which we named this situation as saturation, i.e., In an entanglement session written as $\sum_{i=0}^{n} |x_l, y, x_r\rangle$, for any fixing x_l and x_r bases, if y covers all possible bases, we then say that the part y in the entanglement is in saturation. This concept is important for generating auto-proof, which will be discussed in Section 2.4.

When $x[i]$ and y are both of type CH, there are two situations. When the two parties belong to the same entanglement session, it is possible that an $\mathcal{X}$ or $\mathcal{R}$ operation de-entangles the session. Since QWhile tracks eventual entanglement. In many cases, HAD type can be viewed as a kind of entanglement. In addition, the QWhile type system make sure that most de-entanglements happen at the end of the algorithm by turning the qubit type to CH $m\infty$, so that after the possible de-entanglement, the only possible application is a measurement.

If $x[i]$ and y are in different entanglement sessions, the situation is similar to when $x[i]$ having Had and y having CH type. It merges the two sessions together through the saturation $x[i]$. For example, in BEA, The quantum Boolean guard computes the following operation $(z < i)@x[i]$ on a Had type variable z (state: $\sum_{k=0}^{2^n} |k\rangle$) and a Nor type factor $x[i]$ (state: $|0\rangle$). The result is an entanglement $\sum_{k=0}^{2^n} |k, k < i\rangle$, where the $x[i]$ position stores the Boolean bit result $k < i$. [4] The algorithm further merges the $|z, x[i]\rangle$ session with a loop body entanglement session y. In this cases, both $|z, x[i]\rangle$ and y are of CH type.

Session Type System. Selected type rules are given in Figure 10. As we have mentioned above, the type system tracks possible eventual entanglement for a group of qubits, which we named a session. The type judgment is given as $\Omega, \mathcal{T} \vdash_g s : \zeta \triangleright \tau$.

Rule TEXP is the type rule for c-mode expressions. The expression a is evaluated and variable x is substituted with the value v in s. TMEA is a similar rule as TEXP, but for M-mode variable. We allow partial measurements in QWhile. Thus, we need to find out a possible entanglement session $\zeta \uplus \zeta'$ that contains y's session (ζ), that is going to disappear because of measurement. Then, we re-calculate the entanglement type information for ζ'.

[4]When $k < i$, $x[i] = 1$ while $\neg(k < i)$, $x[i] = 0$.

TA-NOR
$$\frac{\Omega \vdash \overline{l} : \zeta \qquad [\![a]\!]\,|\overline{d}\rangle = \alpha\,|\overline{d'}\rangle}{\Omega, \mathcal{T}[\zeta \mapsto \bigotimes_{n} \texttt{Nor}\ \overline{d}] \vdash_g \overline{l} \leftarrow a : \zeta \triangleright_g \bigotimes_{n} \texttt{Nor}\ \overline{d'}}$$

TA-HAD
$$\frac{\Omega \vdash \overline{l} : \zeta \qquad \zeta \subseteq \zeta' \qquad (\forall \overline{d}\ .\ [\![a]\!]\,|\overline{d}\rangle = |\overline{d'}\rangle)}{\Omega, \mathcal{T}[\zeta' \mapsto \bigotimes_{n} \texttt{Had}\ \bigcirc] \vdash_g \overline{l} \leftarrow a : \zeta' \triangleright \bigotimes_{n} \texttt{Had}\ \bigcirc}$$

TEXP
$$\frac{\Omega[x \mapsto \texttt{c}], \mathcal{T} \vdash_g s[v/x] : \zeta \triangleright \tau}{\Omega, \mathcal{T} \vdash_g \texttt{let}\ x\ =\ v\ \texttt{in}\ s : \zeta \triangleright \tau}$$

TMEA
$$\frac{\begin{array}{c}\Omega \vdash y : \zeta \qquad n' = \texttt{size}(\zeta') \qquad m' = \texttt{size}(\{\overline{c'} | \overline{c} \cdot \overline{c'} \in \beta \cdot \beta' \wedge \overline{c} = \texttt{val}(x)\}) \\ \Omega[x \mapsto \texttt{q}], \mathcal{T}[\zeta' \mapsto \bigotimes_{n'} \texttt{CH}\ m' \{\overline{c'} | \overline{c} \cdot \overline{c'} \in \beta \cdot \beta' \wedge \overline{c} = \texttt{val}(x)\}] \vdash_{\texttt{c}} s : \zeta'' \triangleright \tau\end{array}}{\Omega, \mathcal{T}[\zeta \uplus \zeta' \mapsto \bigotimes_{n} \texttt{CH}\ m(\beta \cdot \beta')] \vdash_{\texttt{c}} \texttt{let}\ x\ =\ \texttt{measure}(y)\ \texttt{in}\ s : \zeta'' \triangleright \tau}$$

TA-CH
$$\frac{\Omega \vdash \overline{l} : \zeta \qquad \zeta' = \zeta \uplus \zeta_r}{\Omega, \mathcal{T}[\zeta' \mapsto \bigotimes_{n} \texttt{CH}\ k\beta] \vdash_g \overline{l} \leftarrow a : \zeta' \triangleright \bigotimes_{n} \texttt{CH}\ k\{\overline{d'} \cdot \overline{d_r} \mid \overline{d} \cdot \overline{d_r} \in \beta(\downarrow \zeta) \cdot \beta(\downarrow \zeta_r) \wedge [\![a]\!]\,|\overline{d}\rangle = \alpha\,|\overline{d'}\rangle\}}$$

TA-MU
$$\frac{\Omega, \mathcal{T}[\zeta_1 \uplus \zeta_3 \uplus \zeta_2 \uplus \zeta_4 \mapsto \bigotimes_{n} \texttt{CH}\ k\beta_1 \cdot \beta_3 \cdot \beta_2 \cdot \beta_4] \vdash_g s : \zeta_1 \uplus \zeta_3 \uplus \zeta_2 \uplus \zeta_4 \triangleright \bigotimes_{n} \texttt{CH}\ k\beta_1 \cdot \beta_3' \cdot \beta_2' \cdot \beta_4}{\Omega, \mathcal{T}[\zeta_1 \uplus \zeta_2 \uplus \zeta_3 \uplus \zeta_4 \mapsto \bigotimes_{n} \texttt{CH}\ k\beta_1 \cdot \beta_2 \cdot \beta_3 \cdot \beta_4] \vdash_g s : \zeta_1 \uplus \zeta_2 \uplus \zeta_3 \uplus \zeta_4 \triangleright \bigotimes_{n} \texttt{CH}\ k\beta_1 \cdot \beta_2' \cdot \beta_3' \cdot \beta_4}$$

TSEQ-1
$$\frac{\begin{array}{c}\Omega, \mathcal{T} \vdash_g s_1 : \zeta \triangleright \bigotimes_{n} \texttt{Nor}\ \overline{d} \qquad \zeta \cup \zeta' \\ \Omega, \mathcal{T}[\zeta \mapsto \bigotimes_{n} \texttt{Nor}\ \overline{d}] \vdash_g s_2 : \zeta' \triangleright \bigotimes_{n'} \texttt{Nor}\ \overline{d'}\end{array}}{\Omega, \mathcal{T} \vdash_g s_1\ ;\ s_2 : \zeta \uplus \zeta' \triangleright \bigotimes_{n+n'} \texttt{Nor}\ \overline{d} \cdot \overline{d'}}$$

TSEQ-2
$$\frac{\begin{array}{c}\Omega, \mathcal{T} \vdash_g s_1 : \zeta \triangleright \bigotimes_{n} \texttt{CH}\ k\beta \\ \Omega, \mathcal{T}[\zeta \mapsto \bigotimes_{n} \texttt{CH}\ k\beta] \vdash_g s_2 : \zeta' \triangleright \tau\end{array}}{\Omega, \mathcal{T} \vdash_g s_1\ ;\ s_2 : \zeta' \triangleright \tau}$$

TIF
$$\frac{\begin{array}{c}\Omega \vdash b(@x[v]) : \zeta \qquad \mathcal{T}(\zeta) = \bigotimes_{n} \texttt{CH}\ k(\beta_1 \cdot 0 \cup \beta_2 \cdot 1) \\ \zeta \cup \zeta' \qquad \texttt{last}(\zeta) = (x, v, v+1) \qquad \Omega, \mathcal{T} \vdash_{\texttt{q}} s : \zeta' \triangleright \bigotimes_{n'} \texttt{CH}\ k'\beta'\end{array}}{\Omega, \mathcal{T}[\zeta \mapsto \bigotimes_{n} \texttt{CH}\ k\beta] \vdash_g \texttt{if}\ (x[v])\ \{s\} : \zeta \uplus \zeta' \triangleright \bigotimes_{n+n'} \texttt{CH}\ (k \times k')(\beta_1 \cdot 0 \cdot \beta' \cup \beta_2 \cdot 1 \cdot \beta')}$$

TLOOP
$$\frac{\begin{array}{c}v_1 \le v < v_2 \qquad \Omega \vdash f(v) : \texttt{c} \qquad \Omega \vdash b(@x[v]) : \zeta(v) \\ \zeta(v) \cup \zeta'(v) \qquad \texttt{last}(\zeta) = (x, v, v+1) \qquad \Omega, \mathcal{T} \vdash_g \texttt{if}\ (b(@x[v]))\ \{s\} : \zeta(v) \uplus \zeta'(v) \triangleright \tau(v)\end{array}}{\Omega, \mathcal{T} \vdash_g \texttt{for}\ (\texttt{int}\ i\ =\ v_1\ ;\ i < v_2\ ;\ b(@x[i])\ ;\ f(i))\ T(i)\ P(i)\ \{s\} : \zeta(v_2) \uplus \zeta'(v_2) \triangleright \tau(v_2)}$$

TDIS
$$\frac{\Omega \vdash x : \zeta \qquad \zeta' = \zeta \uplus \zeta_r \qquad n' = \texttt{size}(\zeta) \qquad m' = \texttt{size}(\{\mathcal{P}(n') \cdot \overline{d_r} \mid \overline{d} \cdot \overline{d_r} \in \beta(\downarrow \zeta) \cdot \beta(\downarrow \zeta_r)\})}{\Omega, \mathcal{T}[\zeta' \mapsto \bigotimes_{n} \texttt{CH}\ m\beta] \vdash_g \texttt{diffuse}(x) : \zeta' \triangleright \bigotimes_{n} \texttt{CH}\ m'\{\mathcal{P}(n') \cdot \overline{d_r} \mid \overline{d} \cdot \overline{d_r} \in \beta(\downarrow \zeta) \cdot \beta(\downarrow \zeta_r)\}}$$

$\overline{d} \cdot \overline{d'}$: list concatenation. $\zeta \cup \zeta'$: The two sessions are disjoint.
$\beta(\downarrow \zeta)$: Get the position range of ζ in the session and form a new set of bitstrings containing only the positions in the range.
$[\![a]\!]\,|\overline{d}\rangle$: $\mathcal{O}$QASM semantics of interpreting reversible expression a in Figure 18.

Fig. 8. Session Type System

TA-NOR, TA-HAD and TA-CH are rules for quantum assignments with different input types. $[\![a]\!]$ appearing in these rules is a semantic function for interpreting the expressions a. The semantic function takes an expression in Nor type and output a Nor value, i.e., inputting classical values and output classical results. The semantics of $\mathcal{O}$QASM (Figure 18) and the arithmetic language $\mathcal{O}$QIMP is the role model of such semantic function. In TA-HAD, when $\overline{l}$ is in uniform superposition ($\texttt{Had}\ \bigcirc$), for every bit in $\overline{l}$, if the semantic function judges that its global phase keeps in uniformity, .i.e, 1, the output type is still a uniform superposition without entanglement. In TA-CH, the factor $\overline{l}$ that is assigning might be a sub-session ζ of the whole entanglement session ζ', such that $\zeta' = \zeta \cdot \zeta_r$. Here, for every element $\overline{d} \cdot \overline{d_r} \in \beta$, we find out the corresponding part $\overline{d}$ belonging to the session ζ ($\overline{d} \cdot \overline{d_r} \in \beta(\downarrow \zeta) \cdot \beta(\downarrow \zeta_r)$), and updates the $\overline{d}$ in the result type.

Rules TSeq-1 and TSeq-2 describe the type for a sequence operation. If s_1 and s_2 are of type $\texttt{Nor}$ or $\texttt{Had}$ (rule TSeq-1), the output session order can be mutated as long as the two sessions are disjoint. If the two sessions are not disjoint, we only need to keep the type for ζ', since it is obvious that $\zeta \subseteq \zeta'$. If s_1 and s_2 are of type $\texttt{CH}$ (rule TSeq-2), we only permit the case when $\zeta \subseteq \zeta'$ for simplicity. It has no technical difficulty to allow τ to be a list and two entanglement sessions to bind together, but it makes the type system a lot messier, and there is no current algorithms that require the modification of two distinct entanglement sessions inside a conditional block. On the other hand, if two distinct entanglement sessions live in a conditional block, the block can always be split into two different conditionals with the same Boolean guard.

Rule $\texttt{TIF}$ describes the type for conditionals when the Boolean guard $b(@x[v])$ having type $\texttt{CH}$, and $x[v]$ is the result bit storing the Boolean evaluation result. The result type of such conditionals is an $\texttt{CH}$ type by merging the session of $b(@x[v])$ into the entanglement session. Assume that the $\texttt{CH}$ bases for $b(@x[v])$ are $\beta_1 \cdot 0$ and $\beta_2 \cdot 1$, meaning that we can split nicely for all possible bases of a quantum state to two different sets where the last bit, which represents the $x[v]$ position, is 0 and 1. For the 0 set, we extend $\beta_1 \cdot 0$ to $\beta_1 \cdot 0 \cdot \beta$ by doing a cross product of the elements in set $\beta_1 \cdot 0$ and set β. For the 1 set, the cross product is applied on set $\beta_1 \cdot 0$ and set β', where β' is the result type bases of the body statement s. It is worth noting that by the subtyping relation in Figure 9, any type can be turned to a $\texttt{CH}$ type. When the Boolean guard has type $\texttt{Nor}$, it is no more than a classical conditional. When the Boolean guard has type $\texttt{Had}$, its behavior is similar to the $\texttt{CH}$ case.

Rule $\texttt{TLOOP}$ describes the type for a $\texttt{for}$-loop. It is a generalization of the conditional case when the Boolean guard $b(@x[i])$ having type $\texttt{CH}$. In the type rule, we pick a number v to represent variable i, and check if a single loop step $\texttt{if}\ (b(@x[v]))\ \{s\}$ is well typed. If so, we can conclude that when we replace v by v_2, the $\texttt{for}$-loop is type checked.

Rule $\texttt{TDIS}$ types a diffusion operator. The rule finds the right part of x in the session ζ'. For every right session bitstring $\overline{d}$ in $\overline{d_l} \cdot \overline{d} \cdot \overline{d_r}$, it generates a set of new type sequence by replacing $\overline{d}$ with elements in the power set $\mathscr{P}(n')$, where n' is the bit length of $\overline{d}$. Here, we need to compute the $\texttt{size}$ of the new bitstring set as m'. Sometimes, this computation can be hard, but for most quantum algorithms, depending on the session data structures, the $\texttt{size}$ can be computed effectively.

1.4 Logic Proof System

The reason of having the session type system in Figure 10 is to enable the proof system given in Figure 11. Every proof rule is a structure as $\Omega \vdash_g \{T\}\ \{P\}\ s\ \{T'\}\ \{Q\}$, where g and Ω are the type entities mentioned in Section 2.3. T and T' are the pre- and post- type predicates for the statement s, meaning that there is type environments $\mathcal{T}$ and $\mathcal{T}'$, such that $\mathcal{T} \models T, \mathcal{T}' \models T', g, \Omega, \mathcal{T} \vdash s : \zeta \triangleright \tau$, and $(\zeta \mapsto \tau) \in \mathcal{T}'$. We denote $(\mathcal{T}, \mathcal{T}') \models (T, s, T') : \zeta \triangleright \tau$ as the property described above. P and Q are the pre- and post- Hoare conditions for statement s.

The proof system is an imitation of the classical Hoare Logic array theory. We view the three different quantum state forms in Figure 13 as arrays with elements in different forms, and use the session types to guide the occurrence of a specific form at a time. Sessions, like the array variables in the classical Hoare Logic theory, represent the stores of quantum states. The state changes are

PA-NOR
$$\frac{(\mathcal{T},\mathcal{T}') \models_g (T, \bar{l} \leftarrow a, T') : \zeta \triangleright \bigotimes_n \texttt{Nor}\ \bar{d}}{\Omega \vdash_g \{T\}\ \{P[[\![a]\!]\zeta/\zeta]\}\ \bar{l} \leftarrow a\ \{T'\}\ \{P\}}$$

PA-CH
$$\frac{(\mathcal{T},\mathcal{T}') \models_g (T, \bar{l} \leftarrow a, T') : \zeta \uplus \zeta' \triangleright \bigotimes_n \texttt{CH}\ m\beta \qquad \mathcal{T}(\bar{l}) = \zeta}{\Omega \vdash_g \{T\}\ \{P[\forall k < m.\ [\![a]\!](\zeta[k])/\zeta[k]]\}\ \bar{l} \leftarrow a\ \{T'\}\ \{P\}}$$

P-MEA
$$\frac{\begin{array}{c}(\mathcal{T}[\forall \zeta'\ .\ \zeta \uplus \zeta' \mapsto \bigotimes_{n+n'} \texttt{CH}\ (m \times m')(\beta \cdot \beta')], \mathcal{T}'[\forall \zeta'\ .\ \zeta' \mapsto \bigotimes_{n'} \texttt{CH}\ m'\beta']) \models_{\mathsf{c}} (T, y, T'') : \zeta \triangleright \bigotimes_n \texttt{CH}\ m\beta \\ v < m \qquad \Omega[x \mapsto M, y \mapsto \bot] \vdash_{\mathsf{c}} \{T''\}\ \{P[(\texttt{as}^2(\zeta[v]), \texttt{bs}(\zeta[v]))/x, \bot/\zeta]\}\ s\ \{T'\}\ \{Q\}\end{array}}{\Omega \vdash_{\mathsf{c}} \{T\}\ \{P\}\ \texttt{let}\ x\ =\ \texttt{measure}(y)\ \texttt{in}\ s\ \{T'\}\ \{Q\}}$$

P-IF
$$\frac{\begin{array}{c}(\mathcal{T},\mathcal{T}') \models_{\mathsf{q}} (T, s, T') : \zeta' \triangleright \bigotimes_{n'} \texttt{CH}\ m'\beta' \\ \Omega \vdash b(@x[v]) : \zeta \uplus [(x, v, v+1)] \qquad \mathcal{T}(\zeta \uplus [(x, v, v+1)]) = \bigotimes_n \texttt{CH}\ 2m(\beta_1 \cdot 0 \cup \beta_2 \cdot 1)\end{array}}{\Omega \vdash_g \{T\}\ \{P[(\zeta \uplus 0 \uplus \zeta') \texttt{++} (\zeta \uplus 1 \uplus [\![s]\!]\zeta')/\zeta \uplus [(x, v, v+1)] \uplus \zeta']\}\ \texttt{if}\ (b(@x[v]))\ \{s\}\ \{T'\}\ \{P\}}$$

P-LOOP
$$\frac{\begin{array}{c}(\mathcal{T},\mathcal{T}') \models_g (T(i), \texttt{if}\ (b(@x[i]))\ \{s\}, T(f(i))) : \zeta \triangleright \tau \\ \Omega \vdash_g \{T(i)\}\ \{P(i)\}\ \texttt{if}\ (x[i])\ \{s\}\ \{T(f(i))\}\ \{P(f(i))\}\end{array}}{\Omega \vdash_g \{T(a_1)\}\ \{P(a_1)\}\ \texttt{for}\ (\texttt{int}\ i\ =\ a_1\ ;\ i < a_2\ ;\ x[i]\ ;\ f(i))\ T(i)\ P(i)\ \{s\}\ \{T(a_2)\}\ \{P(a_2)\}}$$

P-DIS
$$\frac{\begin{array}{c}(\mathcal{T},\mathcal{T}') \models_g (T, \texttt{diffuse}(x), T') : \zeta \triangleright \bigotimes_{n'} \texttt{CH}\ m'\beta' \\ \mathcal{T}(x) = \{\zeta : \bigotimes_{n'} \texttt{CH}\ m\beta\} \qquad \zeta = \zeta' \uplus (x, 0, \Sigma(x)) \qquad \beta_1 \cdot \beta_2 = \beta(\downarrow (x, 0, \Sigma(x))) \cdot \beta(\downarrow \zeta')\end{array}}{\Omega \vdash_g \{T\}\ \{P[\texttt{dis}(n', \zeta, \beta, \beta_1, \beta_2)/\zeta]\}\ \texttt{diffuse}(x)\ \{T'\}\ \{P\}}$$

$\texttt{ln}(\zeta)$: length of ζ $\quad$ $\texttt{as}(\zeta[v])$: the amplitude of $\zeta[v]$ $\quad$ $\texttt{bs}(\zeta[v])$: the base of $\zeta[v]$ $\quad$ ++: array concatenation.

$$\begin{array}{l}\texttt{dis}(n, \zeta, \beta, \beta_1, \beta_2) \equiv \{\frac{1}{2^{n-1}}(\sum_k \texttt{as}(\zeta[k]) - \texttt{as}(\zeta[j]))\beta_1[j] \mid \beta_1[j] \in \mathcal{P}(n)\} \\ \qquad\qquad \cup \bigcup_{j \in \beta_2} \{\frac{1}{2^n} \sum_k \texttt{as}(\zeta[k])z \mid z \in \mathcal{P}(n) \wedge (\forall k.\ z \cdot \beta_2[j] \neq \beta[k])\}\end{array}$$

Fig. 9. Selected Proof System Rules

implemented by the substitutions of sessions with expressions containing operation's semantic transitions. The substitutions can happen for a single index session element or the whole session.

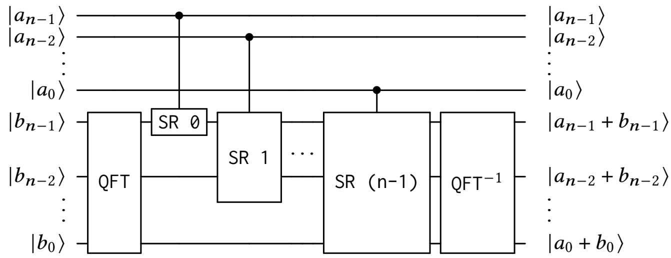
Rule PA-NOR and PA-CH specify the assignment rules. If a session ζ has type $\texttt{Nor}$, it is a singleton array, so the substitution $[\![a]\!]\zeta/\zeta$ means that we substitute the singleton array by a term with the a's application. When ζ has type $\texttt{CH}$, term $\zeta[k]$ refers to each basis state in the entanglement. The assignment is an array map operation that applies a to every element in the array. For example, in Figure 2 line 12, we apply a series of $\texttt{H}$ gates to array x. Its post-condition is $[(x, 0, n)] = \bigotimes_{k=0}^{n} |\Phi(0)\rangle$, where $[(x, 0, n)]$ is the session representing register variable x. Thus, replacing the session $[(x, 0, n)]$ with the $\texttt{H}$ application results in a pre-condition as $\texttt{H}[(x, 0, n)] = \bigotimes_{k=0}^{n} |\Phi(0)\rangle$, which means that $[(x, 0, n)]$ has the state $|0\rangle^n$.

Rule P-MEA is the rule for partial/complete measurement. y's session is ζ, but it might be a part of an entangled session $\zeta \uplus \zeta'$. After the measurement, M-mode x has the measurement result $(\texttt{as}(\zeta[v])^2, \texttt{bs}(\zeta[v]))$ coming from one possible basis state of y (picking a random index v in ζ), $\texttt{as}(\zeta[v])$ is the amplitude and $\texttt{bs}(\zeta[v])$ is the base. We also remove y and its session ζ $(\bot/\zeta)$ in the new pre-condition because it is measured away. The removal means that the entangled session $\zeta \uplus \zeta'$ is replaced by ζ' with the re-computation of the amplitudes and bases for each term.

Rule P-IF deals with a quantum conditional where the Boolean guard $b(@x[v])$ is of type $\bigotimes_n\ \texttt{CH}\ 2m(\beta_1 \cdot 0 \cup \beta_2 \cdot 1)$. The bases are split into two sets $\beta_1 \cdot 0$ and $\beta_2 \cdot 1$, where the last bit represents the base state for the $x[v]$ position. In quantum computing, a conditional is more similar to an assignment, where we create a new array to substitute the current state represented by the session $\zeta \uplus [(x, v, v+1)] \uplus \zeta'$. Here, the new array is given as $(\zeta \uplus 0 \uplus \zeta')$++$(\zeta \uplus 1 \uplus [\![s]\!]\zeta')$, where we double the array: if the $x[v]$ position is 0, we concatenate the current session ζ' for the conditional body, if $x[v] = 1$, we apply $[\![s]\!]$ on the array ζ' and concatenate it to $(\zeta \uplus 1)$.

Rule P-Loop is an initiation of the classical while rule in Hoare Logic with the loop guard possibly having quantum variables. In QWhile, we only has `for-loop` structure and we believe it is enough to specify any current quantum algorithms. For any i, if we can maintain the loop invariant $P(i)$ and $T(i)$ with the post-state $P(f(i))$ and $T(f(i))$ for a single conditional `if` $(x[i])$ $\{s\}$, the invariant is maintained for multiple steps for i from the lower-bound a_1 to the upper bound a_2.

Rule P-DIS proves a diffusion operator `diffuse`(x). The quantum semantics for `diffuse`(x) is $\frac{1}{2^n}(2\sum_{i=0}^{2^n}(\sum_{j=0}\alpha_j)\,|i\rangle - \sum_{j=0}\alpha_j\,|x_j\rangle)$. As an array operation, `diffuse`(x) with the session ζ is an array operation as follows: assume that $\zeta = (x, 0, \Sigma(x)) \uplus \zeta_1$, for every k, if $\zeta[k]$'s value is $\theta_k(\overline{d_x} \cdot \overline{d_1})$, for any bitstring z in $\mathcal{P}(\Sigma(x))$, if $z \cdot \overline{d_1}$ is not a base for $\zeta[j]$ for any j, then the state is $\frac{1}{2^{n-1}}\sum_{k=0}\theta_k(z \cdot \overline{d_1})$; if the base of $\zeta[j]$ is $z \cdot \overline{d_1}$, then the state for $\zeta[j]$ is $\frac{1}{2^{n-1}}(\sum_{k=0}\theta_k) - \theta_j(z \cdot \overline{d_1})$.



(a) Quantum circuit

```
Fixpoint rz_adder' (a b:var) (n:ℕ)
  := match n with
     | 0 ⇒ ID (a,0)
     | S m ⇒ CU (a,m) (SR m b);
             rz_adder' a b m
     end.
Definition rz_adder (a b:var) (n:ℕ)
  := Rev a ; Rev b ; QFT b ;
     rz_adder' a b n;
     QFT⁻¹ b; Rev b ; Rev a.
```

(b) OQASM metaprogram (in Coq)

Fig. 10. Example OQASM program: QFT-based adder

Bit	b	::=	$0 \mid 1$
Natural number	n	$\in$	$\mathbb{N}$
Real	r	$\in$	$\mathbb{R}$
Phase	$\alpha(r)$	::=	$e^{2\pi i r}$
Basis	τ	::=	$\texttt{Nor} \mid \texttt{Phi}\ n$
Unphased qubit	$\overline{q}$	::=	$\|b\rangle \ \mid\ \|\Phi(r)\rangle$
Qubit	q	::=	$\alpha(r)\overline{q}$
State (length d)	φ	::=	$q_1 \otimes q_2 \otimes \cdots \otimes q_d$

Fig. 11. OQASM state syntax

2 OQASM: AN ASSEMBLY LANGUAGE FOR QUANTUM ORACLES

We designed OQASM to be able to express efficient quantum oracles that can be easily tested and, if desired, proved correct. OQASM operations leverage both the standard computational basis and an alternative basis connected by the quantum Fourier transform (QFT). OQASM's type system tracks the bases of variables in OQASM programs, forbidding operations that would introduce entanglement. OQASM states are therefore efficiently represented, so programs can be effectively tested and are simpler to verify and analyze. In addition, OQASM uses *virtual qubits* to support *position shifting operations*, which support arithmetic operations without introducing extra gates during translation. All of these features are novel to quantum assembly languages.

This section presents OQASM states and the language's syntax, semantics, typing, and soundness results. As a running example, we use the QFT adder [?] shown in Figure 12. The Coq function `rz_adder` generates an OQASM program that adds two natural numbers `a` and `b`, each of length `n` qubits.

2.1 OQASM States

An OQASM program state is represented according to the grammar in Figure 13. A state φ of d qubits is a length-d tuple of qubit values q; the state models the tensor product of those values. This means that the size of φ is $O(d)$ where d is the number of qubits. A d-qubit state in a language like SQIR is represented as a length 2^d vector of complex numbers, which is $O(2^d)$ in the number of qubits. Our linear state representation is possible because applying any well-typed OQASM program on any well-formed OQASM state never causes qubits to be entangled.

A qubit value q has one of two forms $\overline{q}$, scaled by a global phase $\alpha(r)$. The two forms depend on the *basis* τ that the qubit is in—it could be either `Nor` or `Phi`. A `Nor` qubit has form $|b\rangle$ (where

$$
\begin{array}{llll}
\text{Position} & p & ::= & (x, n) \qquad \text{Nat. Num } n \qquad \text{Variable } x \\
\text{Instruction} & \iota & ::= & \texttt{ID}\ p \mid \texttt{X}\ p \mid \texttt{RZ}^{[-1]}\ n\ p \mid \iota\ ;\ \iota \\
 & & \mid & \texttt{SR}^{[-1]}\ n\ x \mid \texttt{QFT}^{[-1]}\ n\ x \mid \texttt{CU}\ p\ \iota \\
 & & \mid & \texttt{Lshift}\ x \mid \texttt{Rshift}\ x \mid \texttt{Rev}\ x
\end{array}
$$

Fig. 12. OQASM syntax. For an operator OP, $\texttt{OP}^{[-1]}$ indicates that the operator has a built-in inverse available.

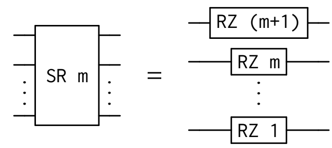


Fig. 13. SR unfolds to a series of RZ instructions

$b \in \{0, 1\}$), which is a computational basis value. A Phi qubit has form $|\Phi(r)\rangle = \frac{1}{\sqrt{2}}(|0\rangle + \alpha(r)\,|1\rangle)$, which is a value of the (A)QFT basis. The number n in Phi n indicates the precision of the state φ. As shown by **?**, arithmetic on the computational basis can sometimes be more efficiently carried out on the QFT basis, which leads to the use of quantum operations (like QFT) when implementing circuits with classical input/output behavior.

2.2 OQASM Syntax, Typing, and Semantics

[Liyi: add RZ gate back]

Figure 14 presents OQASM's syntax. An OQASM program consists of a sequence of instructions ι. Each instruction applies an operator to either a variable x, which represents a group of qubits, or a *position* p, which identifies a particular offset into a variable x.

The instructions in the first row correspond to simple single-qubit quantum gates—ID p, X p, and $\texttt{RZ}^{[-1]}\ n\ p$ —and instruction sequencing. The instructions in the next row apply to whole variables: QFT $n\ x$ applies the AQFT to variable x with n-bit precision and $\texttt{QFT}^{-1}\ n\ x$ applies its inverse. If n is equal to the size of x, then the AQFT operation is exact. $\texttt{SR}^{[-1]}\ n\ x$ applies a series of RZ gates (Figure 15). Operation CU $p\ \iota$ applies instruction ι *controlled* on qubit position p. All of the operations in this row—SR, QFT, and CU—will be translated to multiple SQIR gates. Function `rz_adder` in Figure 12(b) uses many of these instructions; e.g., it uses QFT and $\texttt{QFT}^{-1}$ and applies CU to the mth position of variable a to control instruction `SR m b`.

In the last row of Figure 14, instructions Lshift x, Rshift x, and Rev x are *position shifting operations*. Assuming that x has d qubits and x_k represents the k-th qubit state in x, Lshift x changes the k-th qubit state to $x_{(k+1)\%d}$, Rshift x changes it to $x_{(k+d-1)\%d}$, and Rev changes it to x_{d-1-k}. In our implementation, shifting is *virtual* not physical. The OQASM translator maintains a logical map of variables/positions to concrete qubits and ensures that shifting operations are no-ops, introducing no extra gates.

Other quantum operations could be added to OQASM to allow reasoning about a larger class of quantum programs, while still guaranteeing a lack of entanglement. In **??**, we show how OQASM can be extended to include the Hadamard gate H, z-axis rotations Rz, and a new basis Had to reason directly about implementations of QFT and AQFT. However, this extension compromises the property of type reversibility (Theorem 3.5, Section 3.3), and we have not found it necessary in oracles we have developed.

Typing. In OQASM, typing is with respect to a *type environment* Ω and a predefined *size environment* Σ, which map OQASM variables to their basis and size (number of qubits), respectively. The typing judgment is written $\Sigma; \Omega \vdash \iota \triangleright \Omega'$ which states that ι is well-typed under Ω and Σ, and transforms the variables' bases to be as in Ω' (Σ is unchanged). **[Liyi: good?]** Σ is fixed because the number of qubits in an execution is always fixed. It is generated in the high level language compiler, such as OQIMP in **??**. The algorithm generates Σ by taking an OQIMP program and scanning through all the variable initialization statements. Select type rules are given in Figure 16; the rules not shown (for ID, Rshift, Rev, $\texttt{RZ}^{-1}$, and $\texttt{SR}^{-1}$) are similar.

$$\textrm{X}\quad \frac{\Omega(x) = \texttt{Nor} \qquad n < \Sigma(x)}{\Sigma; \Omega \vdash \texttt{X}\ (x, n) \triangleright \Omega}
\qquad
\textrm{RZ}\quad \frac{\Omega(x) = \texttt{Nor} \qquad n < \Sigma(x)}{\Sigma; \Omega \vdash \texttt{RZ}\ q\ (x, n) \triangleright \Omega}$$

$$\textrm{SR}\quad \frac{\Omega(x) = \texttt{Phi}\ n \qquad m < n}{\Sigma; \Omega \vdash \texttt{SR}\ m\ x \triangleright \Omega}
\qquad
\textrm{QFT}\quad \frac{\Omega(x) = \texttt{Nor} \qquad n \leq \Sigma(x)}{\Sigma; \Omega \vdash \texttt{QFT}\ n\ x \triangleright \Omega[x \mapsto \texttt{Phi}\ n]}$$

$$\textrm{RQFT}\quad \frac{\Omega(x) = \texttt{Phi}\ n \qquad n \leq \Sigma(x)}{\Sigma; \Omega \vdash \texttt{QFT}^{-1}\ n\ x \triangleright \Omega[x \mapsto \texttt{Nor}]}
\qquad
\textrm{CU}\quad \frac{\Omega(x) = \texttt{Nor} \qquad \texttt{fresh}\ (x, n)\ \iota \qquad \Sigma; \Omega \vdash \iota \triangleright \Omega \qquad \texttt{neutral}(\iota)}{\Sigma; \Omega \vdash \texttt{CU}\ (x, n)\ \iota \triangleright \Omega}$$

$$\textrm{LSH}\quad \frac{\Omega(x) = \texttt{Nor}}{\Sigma; \Omega \vdash \texttt{Lshift}\ x \triangleright \Omega}
\qquad
\textrm{SEQ}\quad \frac{\Sigma; \Omega \vdash \iota_1 \triangleright \Omega' \qquad \Sigma; \Omega' \vdash \iota_2 \triangleright \Omega''}{\Sigma; \Omega \vdash \iota_1\ ;\ \iota_2 \triangleright \Omega''}$$

Fig. 14. Select 𝔒QASM typing rules

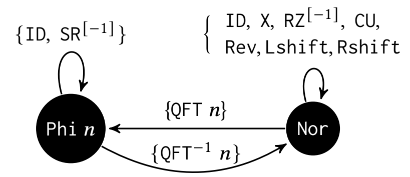


Fig. 15. Type rules' state machine

The type system enforces three invariants. First, it enforces that instructions are well-formed, meaning that gates are applied to valid qubit positions (the second premise in X) and that any control qubit is distinct from the target(s) (the `fresh` premise in CU). This latter property enforces the quantum *no-cloning rule*. For example, we can apply the CU in `rz_adder'` (Figure 12) because position `a,m` is distinct from variable `b`.

Second, the type system enforces that instructions leave affected qubits in a proper basis (thereby avoiding entanglement). The rules implement the state machine shown in Figure 17. For example, $\texttt{QFT}\ n$ transforms a variable from `Nor` to $\texttt{Phi}\ n$ (rule QFT), while $\texttt{QFT}^{-1}\ n$ transforms it from $\texttt{Phi}\ n$ back to `Nor` (rule RQFT). Position shifting operations are disallowed on variables x in the `Phi` basis because the qubits that make up x are internally related (see Definition 3.1) and cannot be rearranged. Indeed, applying a `Lshift` and then a $\texttt{QFT}^{-1}$ on x in `Phi` would entangle x's qubits.

Third, the type system enforces that the effect of position shifting operations can be statically tracked. The `neutral` condition of CU requires that any shifting within ι is restored by the time it completes. For example, $\texttt{CU}\ p\ (\texttt{Lshift}\ x)\ ;\ \texttt{X}\ (x, 0)$ is not well-typed, because knowing the final physical position of qubit $(x, 0)$ would require statically knowing the value of p. On the other hand, the program $\texttt{CU}\ c\ (\texttt{Lshift}\ x\ ;\ \texttt{X}\ (x, 0)\ ;\ \texttt{Rshift}\ x)\ ;\ \texttt{X}\ (x, 0)$ is well-typed because the effect of the `Lshift` is "undone" by an `Rshift` inside the body of the CU.

Semantics. We define the semantics of an 𝔒QASM program as a partial function $[\![\,]\!]$ from an instruction ι and input state φ to an output state φ', written $[\![\iota]\!]\varphi = \varphi'$, shown in Figure 18.

Recall that a state φ is a tuple of d qubit values, modeling the tensor product $q_1 \otimes \cdots \otimes q_d$. The rules implicitly map each variable x to a range of qubits in the state, e.g., $\varphi(x)$ corresponds to some sub-state $q_k \otimes \cdots \otimes q_{k+n-1}$ where $\Sigma(x) = n$. Many of the rules in Figure 18 update a *portion* of a state. We write $\varphi[(x, i) \mapsto q_{(x,i)}]$ to update the i-th qubit of variable x to be the (single-qubit) state $q_{(x,i)}$, and $\varphi[x \mapsto q_x]$ to update variable x according to the qubit *tuple* q_x. $\varphi[(x, i) \mapsto \uparrow q_{(x,i)}]$ and $\varphi[x \mapsto \uparrow q_x]$ are similar, except that they also accumulate the previous global phase of $\varphi(x, i)$ (or $\varphi(x)$). We use $\downarrow$ to convert a qubit $\alpha(b)\overline{q}$ to an unphased qubit $\overline{q}$.

Function `xg` updates the state of a single qubit according to the rules for the standard quantum gate X. `cu` is a conditional operation depending on the `Nor`-basis qubit (x, i). **[Liyi: good?]** RZ (or $\texttt{RZ}^{-1}$) is an z-axis phase rotation operation. Since it applies to `Nor`-basis, it applies a global phase.

$$
\begin{array}{lll}
[\![\texttt{ID}\ p]\!]\varphi & = \varphi & \\
[\![\texttt{X}\ (x,i)]\!]\varphi & = \varphi[(x,i) \mapsto \uparrow \texttt{xg}(\downarrow \varphi(x,i))] & \texttt{where}\ \texttt{xg}(|0\rangle) = |1\rangle \qquad \texttt{xg}(|1\rangle) = |0\rangle \\
[\![\texttt{CU}\ (x,i)\ \iota]\!]\varphi & = \texttt{cu}(\downarrow \varphi(x,i), \iota, \varphi) & \texttt{where}\ \texttt{cu}(|0\rangle, \iota, \varphi) = \varphi \qquad \texttt{cu}(|1\rangle, \iota, \varphi) = [\![\iota]\!]\varphi \\
[\![\texttt{RZ}\ m\ (x,i)]\!]\varphi & = \varphi[(x,i) \mapsto \uparrow \texttt{rz}(m, \downarrow \varphi(x,i))] & \texttt{where}\ \texttt{rz}(m,|0\rangle) = |0\rangle \qquad \texttt{rz}(m,|1\rangle) = \alpha(\frac{1}{2^m})\,|1\rangle \\
[\![\texttt{RZ}^{-1}\ m\ (x,i)]\!]\varphi & = \varphi[(x,i) \mapsto \uparrow \texttt{rrz}(m, \downarrow \varphi(x,i))] & \texttt{where}\ \texttt{rrz}(m,|0\rangle) = |0\rangle \qquad \texttt{rrz}(m,|1\rangle) = \alpha(-\frac{1}{2^m})\,|1\rangle \\
[\![\texttt{SR}\ m\ x]\!]\varphi & = \varphi[\forall i \le m.\ (x,i) \mapsto \uparrow |\Phi(r_i + \frac{1}{2^{m-i+1}})\rangle] & \texttt{when}\ \ \downarrow \varphi(x,i) = |\Phi(r_i)\rangle \\
[\![\texttt{SR}^{-1}\ m\ x]\!]\varphi & = \varphi[\forall i \le m.\ (x,i) \mapsto \uparrow |\Phi(r_i - \frac{1}{2^{m-i+1}})\rangle] & \texttt{when}\ \ \downarrow \varphi(x,i) = |\Phi(r_i)\rangle \\
[\![\texttt{QFT}\ n\ x]\!]\varphi & = \varphi[x \mapsto \uparrow \texttt{qt}(\Sigma(x), \downarrow \varphi(x), n)] & \texttt{where}\ \texttt{qt}(i, |y\rangle, n) = \bigotimes_{k=0}^{i-1}(|\Phi(\frac{y}{2^{n-k}})\rangle) \\
[\![\texttt{QFT}^{-1}\ n\ x]\!]\varphi & = \varphi[x \mapsto \uparrow \texttt{qt}^{-1}(\Sigma(x), \downarrow \varphi(x), n)] & \\
[\![\texttt{Lshift}\ x]\!]\varphi & = \varphi[x \mapsto \texttt{pm}_l(\varphi(x))] & \texttt{where}\ \texttt{pm}_l(q_0 \otimes q_1 \otimes \cdots \otimes q_{n-1}) = q_{n-1} \otimes q_0 \otimes q_1 \otimes \cdots \\
[\![\texttt{Rshift}\ x]\!]\varphi & = \varphi[x \mapsto \texttt{pm}_r(\varphi(x))] & \texttt{where}\ \texttt{pm}_r(q_0 \otimes q_1 \otimes \cdots \otimes q_{n-1}) = q_1 \otimes \cdots \otimes q_{n-1} \otimes q_0 \\
[\![\texttt{Rev}\ x]\!]\varphi & = \varphi[x \mapsto \texttt{pm}_a(\varphi(x))] & \texttt{where}\ \texttt{pm}_a(q_0 \otimes \cdots \otimes q_{n-1}) = q_{n-1} \otimes \cdots \otimes q_0 \\
[\![\iota_1; \iota_2]\!]\varphi & = [\![\iota_2]\!]([\![\iota_1]\!]\varphi) &
\end{array}
$$

$$
\begin{array}{l}
\downarrow \alpha(b)\overline{q} = \overline{q} \qquad \downarrow (q_1 \otimes \cdots \otimes q_n) = \downarrow q_1 \otimes \cdots \otimes \downarrow q_n \\
\varphi[(x,i) \mapsto \uparrow \overline{q}] = \varphi[(x,i) \mapsto \alpha(b)\overline{q}] \qquad \texttt{where}\ \ \varphi(x,i) = \alpha(b)\overline{q_i} \\
\varphi[(x,i) \mapsto \uparrow \alpha(b_1)\overline{q}] = \varphi[(x,i) \mapsto \alpha(b_1 + b_2)\overline{q}] \qquad \texttt{where}\ \ \varphi(x,i) = \alpha(b_2)\overline{q_i} \\
\varphi[x \mapsto q_x] = \varphi[\forall i < \Sigma(x).\ (x,i) \mapsto q_{(x,i)}] \\
\varphi[x \mapsto \uparrow q_x] = \varphi[\forall i < \Sigma(x).\ (x,i) \mapsto \uparrow q_{(x,i)}]
\end{array}
$$

Fig. 16. $\mathcal{O}$QASM semantics

By Theorem 3.4, when we compiles it to SQIR, the global phase might be turned to a local one. For example, to prepare the state $\sum_{j=0}^{2^n}(-i)^x\,|x\rangle$ [?], we apply a series of Hadamard gates following by several controlled-RZ gates on x, where the controlled-RZ gates are definable by $\mathcal{O}$QASM. SR (or $\texttt{SR}^{-1}$) applies an $m+1$ series of RZ (or $\texttt{RZ}^{-1}$) rotations where the i-th rotation applies a phase of $\alpha(\frac{1}{2^{m-i+1}})$ (or $\alpha(-\frac{1}{2^{m-i+1}})$). qt applies an approximate quantum Fourier transform; $|y\rangle$ is an abbreviation of $|b_1\rangle \otimes \cdots \otimes |b_i\rangle$ (assuming $\Sigma(y) = i$) and n is the degree of approximation. If $n = i$, then the operation is the standard QFT. Otherwise, each qubit in the state is mapped to $|\Phi(\frac{y}{2^{n-k}})\rangle$, which is equal to $\frac{1}{\sqrt{2}}(|0\rangle + \alpha(\frac{y}{2^{n-k}})\,|1\rangle)$ when $k < n$ and $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$ when $n \le k$ (since $\alpha(n) = 1$ for any natural number n). $\texttt{qt}^{-1}$ is the inverse function of qt. Note that the input state to $\texttt{qt}^{-1}$ is guaranteed to have the form $\bigotimes_{k=0}^{i-1}(|\Phi(\frac{y}{2^{n-k}})\rangle)$ because it has type Phi n. $\texttt{pm}_l$, $\texttt{pm}_r$, and $\texttt{pm}_a$ are the semantics for `Lshift`, `Rshift`, and `Rev`, respectively.

2.3 $\mathcal{O}$QASM Metatheory

Soundness. We prove that well-typed $\mathcal{O}$QASM programs are well defined; i.e., the type system is sound with respect to the semantics. We begin by defining the well-formedness of an $\mathcal{O}$QASM state.

Definition 2.1 (Well-formed $\mathcal{O}$QASM state). A state φ is *well-formed*, written $\Sigma; \Omega \vdash \varphi$, iff:

- For every $x \in \Omega$ such that $\Omega(x) = \texttt{Nor}$, for every $k < \Sigma(x)$, $\varphi(x,k)$ has the form $\alpha(r)\,|b\rangle$.
- For every $x \in \Omega$ such that $\Omega(x) = \texttt{Phi}\ n$ and $n \le \Sigma(x)$, there exists a value v such that for every $k < \Sigma(x)$, $\varphi(x,k)$ has the form $\alpha(r)\,|\Phi(\frac{v}{2^{n-k}})\rangle$.[5]

Type soundness is stated as follows; the proof is by induction on ι, and is mechanized in Coq.

[5]Note that $\Phi(x) = \Phi(x+n)$, where the integer n refers to phase $2\pi n$; so multiple choices of v are possible.

$$\texttt{X}\ (x,n) \xrightarrow{\text{inv}} \texttt{X}\ (x,n) \qquad \texttt{SR}\ m\ x \xrightarrow{\text{inv}} \texttt{SR}^{-1}\ m\ x \qquad \texttt{QFT}\ n\ x \xrightarrow{\text{inv}} \texttt{QFT}^{-1}\ n\ x \qquad \texttt{Lshift}\ x \xrightarrow{\text{inv}} \texttt{Rshift}\ x$$

$$\frac{\iota \xrightarrow{\text{inv}} \iota'}{\texttt{CU}\ (x,n)\ \iota \xrightarrow{\text{inv}} \texttt{CU}\ (x,n)\ \iota'} \qquad \frac{\iota_1 \xrightarrow{\text{inv}} \iota_1' \qquad \iota_2 \xrightarrow{\text{inv}} \iota_2'}{\iota_1\ ;\ \iota_2 \xrightarrow{\text{inv}} \iota_2'\ ;\ \iota_1'}$$

Fig. 17. Select OQASM inversion rules

THEOREM 2.2. [OQASM type soundness] If $\Sigma; \Omega \vdash \iota \triangleright \Omega'$ and $\Sigma; \Omega \vdash \varphi$ then there exists φ' such that $[\![\iota]\!]\varphi = \varphi'$ and $\Sigma; \Omega' \vdash \varphi'$.

Algebra. Mathematically, the set of well-formed d-qubit OQASM states for a given Ω can be interpreted as a subset $\mathcal{S}^d$ of a 2^d-dimensional Hilbert space $\mathcal{H}^d$,[6] and the semantics function $[\![\,]\!]$ can be interpreted as a $2^d \times 2^d$ unitary matrix, as is standard when representing the semantics of programs without measurement [?]. Because OQASM's semantics can be viewed as a unitary matrix, correctness properties extend by linearity from $\mathcal{S}^d$ to $\mathcal{H}^d$—an oracle that performs addition for classical Nor inputs will also perform addition over a superposition of Nor inputs. We have proved that $\mathcal{S}^d$ is closed under well-typed OQASM programs.

[Liyi: good?] Given a qubit size map Σ and type environment Ω, the set of OQASM programs that are well-typed with respect to Σ and Ω (i.e., $\Sigma; \Omega \vdash \iota \triangleright \Omega'$) form a algebraic structure $(\{\iota\}, \Sigma, \Omega, \mathcal{S}^d)$, where $\{\iota\}$ defines the set of valid program syntax, such that there exists Ω', $\Sigma; \Omega \vdash \iota \triangleright \Omega'$ for all ι in $\{\iota\}$; $\mathcal{S}^d$ is the set of d-qubit states on which programs $\iota \in \{\iota\}$ are run, and are well-formed ($\Sigma; \Omega \vdash \varphi$) according to Definition 3.1. From the OQASM semantics and the type soundness theorem, for all $\iota \in \{\iota\}$ and $\varphi \in \mathcal{S}^d$, such that $\Sigma; \Omega \vdash \iota \triangleright \Omega'$ and $\Sigma; \Omega \vdash \varphi$, we have $[\![\iota]\!]\varphi = \varphi'$, $\Sigma; \Omega' \vdash \varphi'$, and $\varphi' \in \mathcal{S}^d$. Thus, $(\{\iota\}, \Sigma, \Omega, \mathcal{S}^d)$, where $\{\iota\}$ defines a groupoid.

We can certainly extend the groupoid to another algebraic structure $(\{\iota'\}, \Sigma, \mathcal{H}^d)$, where $\mathcal{H}^d$ is a general 2^d dimensional Hilbert space $\mathcal{H}^d$ and $\{\iota'\}$ is a universal set of quantum gate operations. Clearly, we have $\mathcal{S}^d \subseteq \mathcal{H}^d$ and $\{\iota\} \subseteq \{\iota'\}$, because sets $\mathcal{H}^d$ and $\{\iota'\}$ can be acquired by removing the well-formed ($\Sigma; \Omega \vdash \varphi$) and well-typed ($\Sigma; \Omega \vdash \iota \triangleright \Omega'$) definitions for $\mathcal{S}^d$ and $\{\iota\}$, respectively. $(\{\iota'\}, \Sigma, \mathcal{H}^d)$ is a groupoid because every OQASM operation is valid in a traditional quantum language like SQIR. We then have the following two two theorems to connect OQASM operations with operations in the general Hilbert space:

THEOREM 2.3. $(\{\iota\}, \Sigma, \Omega, \mathcal{S}^d) \subseteq (\{\iota\}, \Sigma, \mathcal{H}^d)$ is a subgroupoid.

THEOREM 2.4. Let $|y\rangle$ be an abbreviation of $\bigotimes_{m=0}^{d-1} \alpha(r_m)\,|b_m\rangle$ for $b_m \in \{0,1\}$. If for every $i \in [0, 2^d)$, $[\![\iota]\!]\,|y_i\rangle = |y_i'\rangle$, then $[\![\iota]\!](\sum_{i=0}^{2^d-1} |y_i\rangle) = \sum_{i=0}^{2^d-1} |y_i'\rangle$.

We prove these theorems as corollaries of the compilation correctness theorem from OQASM to SQIR (**??**). Theorem 3.3 suggests that the space $\mathcal{S}^d$ is closed under the application of any well-typed OQASM operation. Theorem 3.4 says that OQASM oracles can be safely applied to superpositions over classical states.[7]

OQASM programs are easily invertible, as shown by the rules in Figure 19. This inversion operation is useful for constructing quantum oracles; for example, the core logic in the QFT-based subtraction circuit is just the inverse of the core logic in the addition circuit (Figure 19). This allows us to reuse

[6]A *Hilbert space* is a vector space with an inner product that is complete with respect to the norm defined by the inner product. $\mathcal{S}^d$ is a sub*set*, not a sub*space* of $\mathcal{H}^d$ because $\mathcal{S}^d$ is not closed under addition: Adding two well-formed states can produce a state that is not well-formed.

[7]Note that a superposition over classical states can describe *any* quantum state, including entangled states.

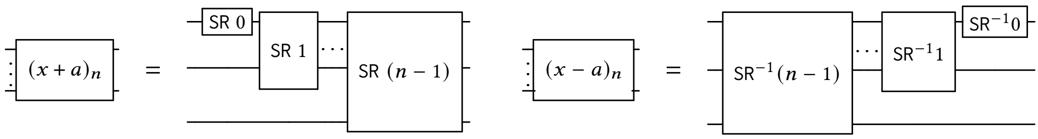


Fig. 18. Addition/subtraction circuits are inverses

the proof of addition in the proof of subtraction. The inversion function satisfies the following properties:

Theorem 2.5. [Type reversibility] For any well-typed program ι, such that $\Sigma; \Omega \vdash \iota \triangleright \Omega'$, its inverse ι', where $\iota \xrightarrow{\text{inv}} \iota'$, is also well-typed and we have $\Sigma; \Omega' \vdash \iota' \triangleright \Omega$. Moreover, $[\![\iota; \iota']\!]\varphi = \varphi$.