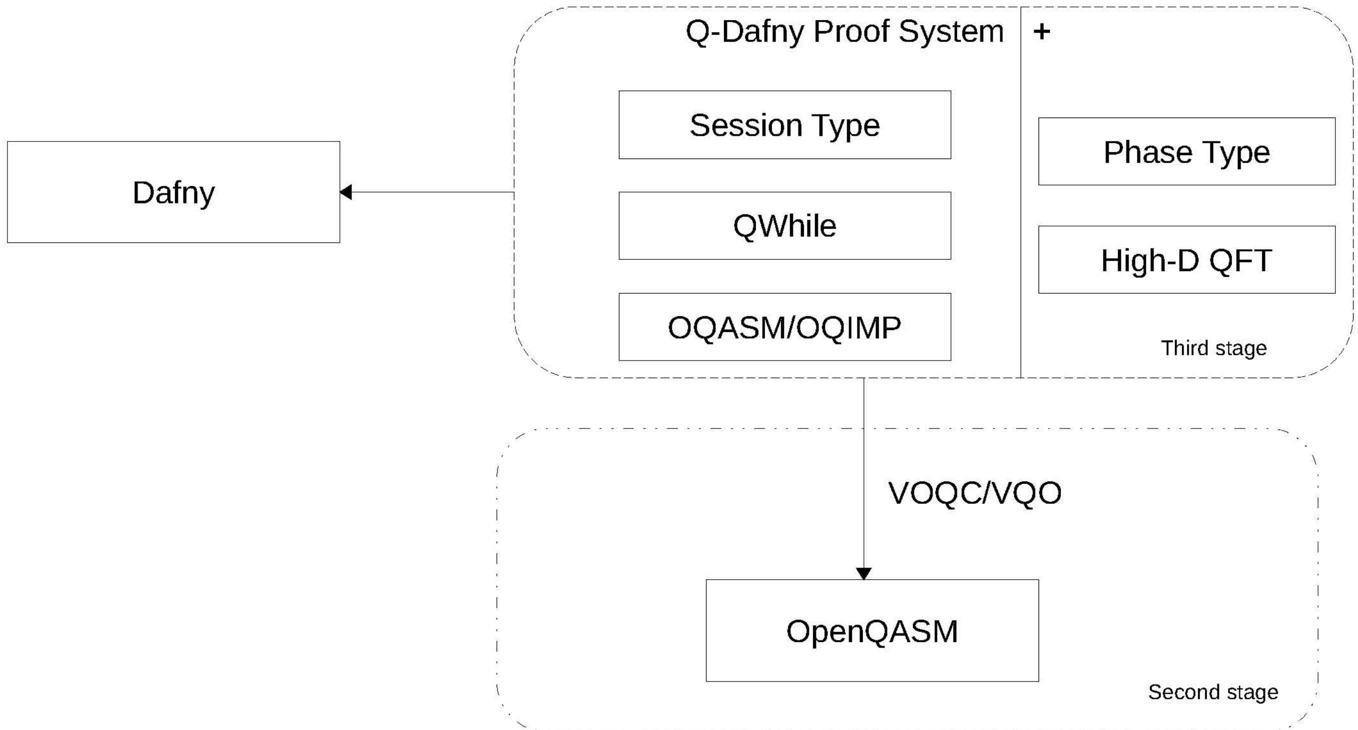
Quantum Dafny Project

Liyi Li

Q-Dafny Introduction

- Verify Quantum Program properties by a classical program logic framework. (quantum part only)
  - Hoare Logic Array Theory
  - Compiling Q-Dafny to Dafny to verify quantum programs.
  - Merging a type system with the Dafny proof system to track quantum state transitions.
  - The system is sound but not complete.
    - First stage, only for QFT; U ; QFT^{-1} structures.
    - Having a rich operation set for U.

Q-Dafny Introduction



Q-Dafny First Stage Compilation

- Using array theories to prove quantum programs.
  - Automated axioms without worrying about sizes.
  - Some quantum state array sizes can be 2^n.
- Define pre-proved axioms.
- Type system transforms quantum state forms from one to the other.
  - Define transformation axioms.
- Compile a Q-Dafny program with the insertions of axioms and transformation functions to Dafny.

Q-Dafny Symtax

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

Q-Dafny Type and State Elements

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

Q-Dafny Example Program – Shor's Algorithm

```
method Shor ( a : int, N : int, n : int, m : int, x : Q[n], y : Q[n] )
 requires (n > 0)
 requires (1 < a < N)
 requires (N < 2^(n-1))
 requires (N^2 < 2^m ≤ 2 * N^2)
 requires (gcd(a, N) == 1)
 requires ( type(x) = Tensor n (Nor 0))
 requires ( type(y) = Tensor n (Nor 0))
 ensures (gcd(N, r) == 1)
 ensures (p.pos ≥ 4 / (PI ^ 2))
{
  x *= H ;
  y *= cl(y+1); //cl can be omitted.
  for (int i = 0; i < n; x[i]; i ++)
    invariant (0 ≤ i ≤ n)
    invariant (saturation(x[0..i]))
    invariant (type(y,x[0..i]) = Tensor n (ch (2^i) {k | j baseof x[0..i] && k = (a^j mod N,j)}))
    //psum(k=b,M,p(k),b(k)) = sum_{k=b}^M p(k)*b(k)
    invariant ((y,x[0..i]) == psum(k=0,2^i,1,(a^k mod N,k)))
  {
    y *= cl(a^(2^i) * y mod N);
  }

 M z := measure(y);//partial measurement, actually measure(y,r) r is the period
 x *= RQFT;
 M p := measure(x); //p.pos and p.base
 var r := post_period(m,p.base) // ∃ t. 2^m * t / r = p.base
}
```

Q-Dafny Example Compilation

- x *= H – transform Nor type to Had Type.
  - [VQO/Shor-compiled.dfy at naturalproof · inQWIRE/VQO · GitHub]
  - H is also a summation for n qubit Hadmard gate for array x.
  - In Dafny
    - Tensor n Nor is implemented as a data type of two ints (one for phase and one for base).
    - Tensor n Had is implemented as an n-array of a pair of an int (representing a phase).
  - The compiled code in Dafny is in the next page.

Q-Dafny Example Compilation

```
method H()
   requires m.Nor?
   requires Wf()
   requires WfNor()
   ensures Wf()
   ensures m.Had?
   ensures m.h.Length == old(m.b.Length) == card
   ensures forall i | 0 <= i < card :: old(m.b[i]) == 0 ==> m.h[i] == 1
   ensures fresh(m.h)
   modifies this
 {
   ghost var tmp := m.b;
   var qs := new int[card];
   var i := 0;
   assert m.Nor?;
   assert qs.Length == m.b.Length == card;
   while i < card
     decreases (card - i)
     invariant m.Nor?
     invariant i <= card
     invariant qs.Length == m.b.Length == card
     invariant forall j | 0 <= j < i :: m.b[j] == 0 ==> qs[j] == 1
     invariant (tmp == m.b);
   {
     qs[i] := if m.b[i] == 0 then 1 else -1;
     i := i + 1;
   }
   assert qs.Length == m.b.Length == card;
   assert (forall i | 0 <= i < card :: m.b[i] == 0 ==> qs[i] == 1);
   m := Had(qs);
 }
```

Q-Dafny Example Compilation

Before the rule, y is Nor, and it is auto-turned to CH 1

```
for (int i = 0; i < n; x[i]; i ++)
  invariant (0 ≤ i ≤ n)
  invariant (saturation(x[0..i]))
  invariant (type(y,x[0..i]) = Tensor n (ch (2^i) {k | j baseof x[0..i] && k = (a^j mod N,j)}))
  //psum(k=b,M,p(k),b(k)) = sum_{k=b}^M p(k)*b(k)
  invariant ((y,x[0..i]) == psum(k=0,2^i,1,(a^k mod N,k)))
{
  y *= cl(a^(2^i) * y mod N);
}
```

Q-Dafny Example Compilation

Tensor n CH m b ➔ data type:

CH(m){ bit_length =n; array_var: [y, x[0..i]]

is_index(x[0..i]); y_array = …; x_array = ... }

element type is a pair of (int * base) : first → phase, second → base

base is a special data-structure that can view a base both as nat or bitstring

```
for (int i = 0; i < n; x[i]; i ++)
  invariant (0 ≤ i ≤ n)
  invariant (saturation(x[0..i]))
  invariant (type(y,x[0..i]) = Tensor n (ch (2^i) {k | j baseof x[0..i] && k = (a^j mod N,j)}))
  //psum(k=b,M,p(k),b(k)) = sum_{k=b}^M p(k)*b(k)
  invariant ((y,x[0..i]) == psum(k=0,2^i,1,(a^k mod N,k)))
{
  y *= cl(a^(2^i) * y mod N);
}
```

Q-Dafny Example Compilation

saturation means that x[0..i]
is treated as index in the datatype

```
for (int i = 0; i < n; x[i]; i ++)
  invariant (0 ≤ i ≤ n)
  invariant (saturation(x[0..i]))
  invariant (type(y,x[0..i]) = Tensor n (ch (2^i) {k | j baseof x[0..i] && k = (a^j mod N,j)}))
  //psum(k=b,M,p(k),b(k)) = sum_{k=b}^M p(k)*b(k)
  invariant ((y,x[0..i]) == psum(k=0,2^i,1,(a^k mod N,k)))
{
  y *= cl(a^(2^i) * y mod N);
}
```

Q-Dafny Example Compilation

At first, x[0..i] is empty because i is 0.

```
for (int i = 0; i < n; x[i]; i ++)
  invariant (0 ≤ i ≤ n)
  invariant (saturation(x[0..i]))
  invariant (type(y,x[0..i]) = Tensor n (ch (2^i) {k | j baseof x[0..i] && k = (a^j mod N,j)}))
  //psum(k=b,M,p(k),b(k)) = sum_{k=b}^M p(k)*b(k)
  invariant ((y,x[0..i]) == psum(k=0,2^i,1,(a^k mod N,k)))
{
  y *= cl(a^(2^i) * y mod N);
}
```

Q-Dafny Example Compilation

In each iteration, a new x[i] is added to the end of the array [y,x[0..i]] ➔ push the session to be [y,x[0..i+1]].

```
for (int i = 0; i < n; x[i]; i ++)
  invariant (0 ≤ i ≤ n)
  invariant (saturation(x[0..i]))
  invariant (type(y,x[0..i]) = Tensor n (ch (2^i) {k | j baseof x[0..i] && k = (a^j mod N,j)}))
  //psum(k=b,M,p(k),b(k)) = sum_{k=b}^M p(k)*b(k)
  invariant ((y,x[0..i]) == psum(k=0,2^i,1,(a^k mod N,k)))
{
  y *= cl(a^(2^i) * y mod N);
}
```

Q-Dafny Example Compilation

In each iteration, array size double.
x[0..i+1] if x[i+1]=0, then
val(x[0..i])=val(x[0..i+1]) as an index.
Thus, y[val(x[0..i+1])=y[val(x[0..i])];

```
for (int i = 0; i < n; x[i]; i ++)
  invariant (0 ≤ i ≤ n)
  invariant (saturation(x[0..i]))
  invariant (type(y,x[0..i]) = Tensor n (ch (2^i) {k | j baseof x[0..i] && k = (a^j mod N,j)}))
  //psum(k=b,M,p(k),b(k)) = sum_{k=b}^M p(k)*b(k)
  invariant ((y,x[0..i]) == psum(k=0,2^i,1,(a^k mod N,k)))
{
  y *= cl(a^(2^i) * y mod N);
}
```

Q-Dafny Example Compilation

If x[i]=1, then val(x[0..i+1])=val(x[0..i+1])+2^i
Thus, y(val(x[0..i+1])) = (a^((val(x[0..i])+2^i) mod N

```
for (int i = 0; i < n; x[i]; i ++)
  invariant (0 ≤ i ≤ n)
  invariant (saturation(x[0..i]))
  invariant (type(y,x[0..i]) = Tensor n (ch (2^i) {k | j baseof x[0..i] && k = (a^j mod N,j)}))
  //psum(k=b,M,p(k),b(k)) = sum_{k=b}^M p(k)*b(k)
  invariant ((y,x[0..i]) == psum(k=0,2^i,1,(a^k mod N,k)))
{
  y *= cl(a^(2^i) * y mod N);
}
```

Q-Dafny Example Compilation

- All the above descriptions are formulated as lemmas and axioms and will be inserted to compiled Dafny programs for proof automation.
- In many (almost all near term) quantum algorithms, entanglement (CH) is created by inserting one qubit at a time.
  - The modular multiplication lemmas are special for Shor’s algorithm.
  - Most likely, the other axioms are useful to deal for many cases.

Q-Dafny Next Step and Potential Problems

- The compiled and verified version is in the above link.
- Finish the Q-Dafny to Dafny Compiler.
  - We have found ways of expanding Dafny to Q-Dafny.
  - Basically, Q-Dafny will be a library on top of Dafny.
- Dafny is based on computing weakest pre-condition.
  - Might not be good for Q-Dafny.
  - Maybe strongest post-condition computation will be faster for quantum program verifications.
    - Quantum computation is used for reversing one-way functions which are hard to dealt with in a classical computer.

Q-Dafny Next Step and Potential Problems

- Need some post-quantum stage library (math) functions.
  - Continued fraction.
  - The final stage of phase estimation to compute probability.
  - Currently, we assume properties for these library functions, and try to implement programs capturing these function properties. We then extract the code to C# and running testing in C# (using numerical methods) to test if these property setups are correct.

Q-Dafny Next Step and Potential Problems

- Not know the necessity of proving these functions in Dafny.
  - They have been proved in Coq.
  - Dafny is a proof framework for implementations more than a theoretical one for program specifications.
  - Maybe testing properties in terms of implementations that rely on floating-point numerical methods will be better than proving them by assuming some real number properties.