Hoare Types for Quantum Programming Languages

Kartik Singhal <ks@cs.uchicago.edu>

EPiQC

THE UNIVERSITY OF CHICAGO

Overview

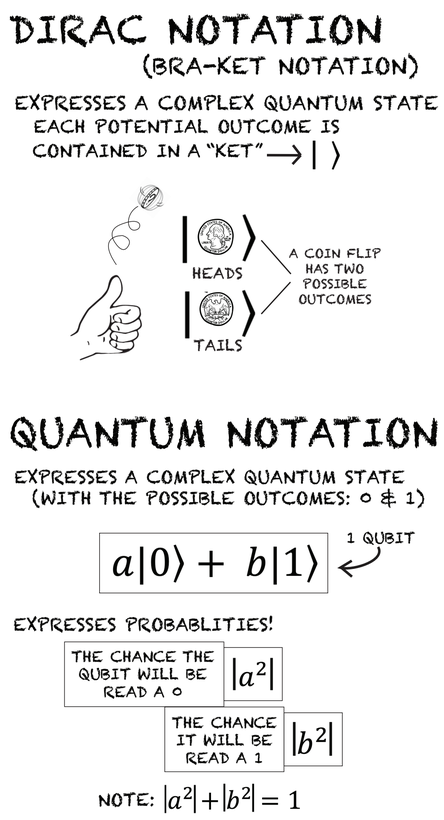
Programming quantum computers will be as **bug-prone** as programming classical computers.

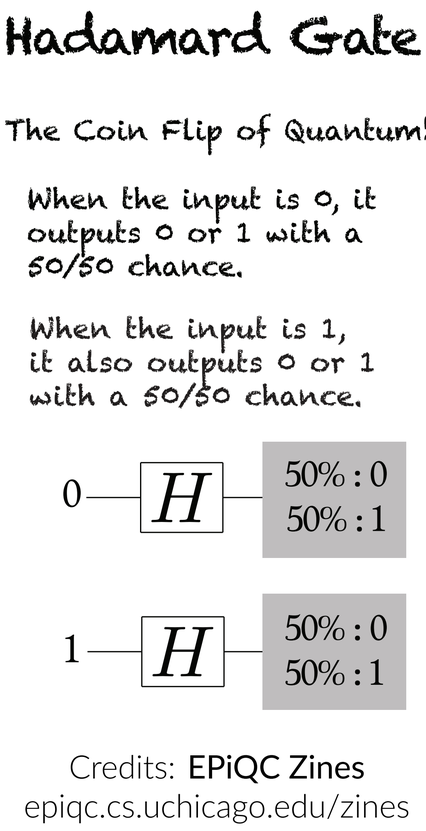
Strong static type systems **eliminate** the possibility of several classes of bugs in classical computing. We need similar assurance of correctness in the quantum paradigm.

We propose **Quantum Hoare Types** that let programmers reason about and enforce certain semantic properties of their quantum programs.

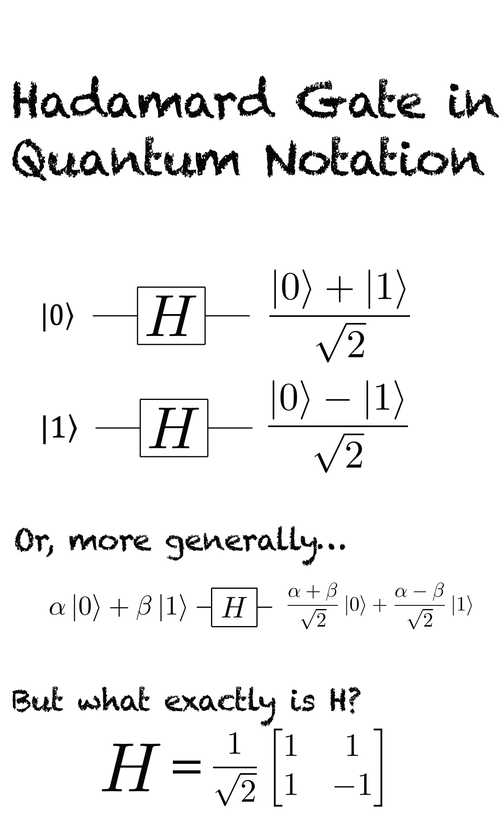
Our appraoch has potential to be a **unified system** for programming, specifying, and reasoning about quantum programs.

Quantum Computing





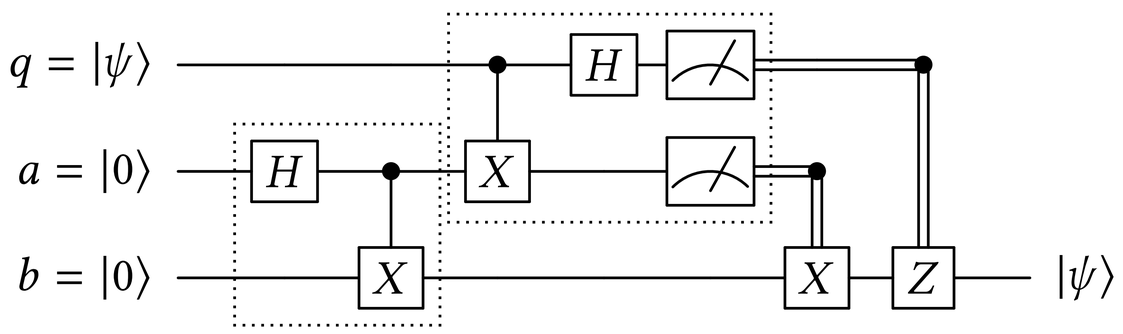
Credits: **EPiQC Zines**
epiqc.cs.uchicago.edu/zines



Example: Quantum Teleportation

Alice is trying to send a message encoded in a qubit q to Bob. There are three broad steps in the protocol:

1. Alice & Bob create entanglement between their two qubits.
2. Alice sends the message.
3. Bob reconstructs the message.




```
teleport : Π|ψ⟩:qbit.{q ↦qbit |ψ⟩ ∧ a ↦qbit |0⟩ ∧ b ↦qbit |0⟩}
      c:Circuit(qbit⊗qbit⊗qbit, qbit) {b ↦qbit |ψ⟩} =
  box (q, a, b) ⇒
  (* 1. Bell state preparation for Alice and Bob *)
  a       ← H a;
  (a, b)  ← CX (a, b);
  (* 2. Bell measurement by Alice *)
  (q, a)  ← CX (q, a);
  q       ← H q;
  x       ← meas q;
  y       ← meas a;
  (* 3. Correction by Bob *)
  (y, b)  ← CX (y, b);
  (x, b)  ← CZ (x, b);
  output b
```

Quantum Hoare Types

A Hoare type [Nanevski et al. 2008] encodes **preconditions** and **post-conditions** in the same spirit as Hoare triples.

$$\{P\}\ \mathtt{x} : A\ \{Q\}$$

A classic example:

$$\mathtt{alloc} : \forall\alpha.\Pi x : \alpha.\{\mathtt{emp}\}\ y : \mathtt{nat}\ \{y \mapsto_\alpha x\}$$

To extend **Hoare Type Theory** to support quantum circuits, we augment its syntax with that of **$\mathcal{Q}$wire** [Paykin et al. 2017] which is a core quantum circuit language that can cooperate with an arbitrary classical host language.

$\mathcal{Q}$wire's circuit language has a wire type W that can be a unit, a bit, a qubit or a tuple of wires:

$$W ::= 1 \mid \mathtt{bit} \mid \mathtt{qbit} \mid W1 \otimes W2$$

The host language (HTT) is augmented with circuit types to support interaction between the two languages and to treat circuits as data:

$$A ::= \ldots \mid \mathtt{Unit} \mid \mathtt{Bool} \mid A \times A \mid \mathtt{Circuit}(W1, W2)$$

A Taste of Verification

The first step of teleportation results in preparation of Bell state:

```
(* P0 (given): {q ↦qbit |ψ⟩ ∧ a ↦qbit |0⟩ ∧ b ↦qbit |0⟩} *)
a       ← H a;
(* P1: {q ↦qbit |ψ⟩ ∧ a ↦qbit 1/√2(|0⟩+|1⟩) ∧ b ↦qbit |0⟩} *)
(a, b) ← CX (a, b);
(* P2: {q ↦qbit |ψ⟩ ∧ (a,b) ↦qbit |ϕ+⟩ *)
```

where $|\phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

Further Work

We look to explore how our types will evolve when we start incorporating **higher-order constructs** in our quantum language such as those demonstrated in languages like Quantum Lambda Calculus [Selinger and Valiron 2006] and Quipper [Green et al. 2013].

Another venue for exploration is to incorporate **more precise types** that can distinguish between qubits in pure classical state vs. those in superposition vs. those in entanglement.