Quantum Hoare Type Theory

Kartik Singhal John Reppy

Department of Computer Science
University of Chicago

Quantum Physics and Logic (QPL) 2020

Quantum programming is inherently imperative and difficult to reason about

In classical programming

Hoare triples are used to reason about state changes.

$$\{P\}\ c\ \{Q\}$$

c is the command to be executed; P, Q are pre and postconditions on state.

In pure functional settings, monads can encapsulate effects.

Can we combine Hoare triples with monadic types?

Yes, thanks to Hoare Type Theory!

For quantum programming?

Outline

Outline

Hoare Types specify pre and postconditions and are very expressive

$$\Delta.\Psi.\{P\}\ \mathtt{x} : A\ \{Q\}$$

P, Q are pre and postconditions (as before)

x is the return value of type A

Δ and Ψ are variable and heap contexts

For example, the type of the `alloc` primitive from HTT:

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

which is a polymorphic function that takes as input x of any type α and returns a new location y of type `nat` after initializing it with x.

Outline

- Motivation
- Background
  - Hoare Type Theory (HTT). Nanevski et al, '07
  - Quantum IO Monad (QIO). Altenkirch & Green, '09
- Quantum Hoare Type Theory (QHTT)
  - Examples
  - Typing Rules
  - Verification
- Ongoing & Future Work
- Conclusion

QIO is a monadic interface for quantum programming implemented in Haskell

QIO monad is indexed by the type of computational result.

```
mkQbit :: Bool → QIO Qbit          -- initialization
applyU :: U → QIO ()               -- apply a unitary
measQbit :: Qbit → QIO Bool        -- measurement
```

Arbitrary unitaries can be defined using:

```
rot :: Qbit → ((Bool, Bool) → C) → U
ifQ :: Qbit → U → U
```

U is monoid with sequencing as its operation
and identity as the neutral element.

Outline

Programming in Quantum Hoare Type Theory

We further index the QIO monad with pre and postconditions to get a *Hoare* monad.

Hello Quantum World:

```
hqw : {emp} r : Bool {emp ∧ Id(r, false)}
    = do q ⇐ mkQbit false;
         measQbit q
```

Quantum Coin Toss:

```
rnd : {emp} r : Bool {emp}
    = do q ⇐ mkQbit false;
         applyU (H q);
         measQbit q
```

But how do we reason about these programs?

Outline

Strongest Postcondition for Initialization

$$x \Leftarrow \texttt{mkQbit}\, M; E$$

HTT uses bidirectional typing for type inference, where
$e \Leftarrow A$ means 'expression e checks against type A', and,
$e \Rightarrow A$ means 'expression e synthesizes the type A'.

$$\frac{\Delta \vdash M \Leftarrow \mathbf{Bool} \qquad \Delta, x : \mathbf{Qbit}; P \circ (x \mapsto \texttt{state}\,(M)) \vdash E \Rightarrow y : B.Q}{\Delta; P \vdash x \Leftarrow \texttt{mkQbit}\, M; E \Rightarrow y : B.(\exists x : \mathbf{Qbit}.Q)}$$

Strongest Postcondition for Measurement

$$x \Leftarrow \texttt{measQbit}\, M; E$$

HTT uses bidirectional typing for type inference, where:
$e \Leftarrow A$ means 'expression e checks against type A', and,
$e \Rightarrow A$ means 'expression e synthesizes the type A'.

$$\frac{\Delta \vdash M \Leftarrow \mathbf{Qbit} \qquad \Delta; \Psi; P \Longrightarrow (M \hookrightarrow -) \qquad \Delta, x : \mathbf{Bool}; P \circ ((M \mapsto -) \multimap \texttt{emp}) \vdash E \Rightarrow y : B.Q}{\Delta; P \vdash x \Leftarrow \texttt{measQbit}\, M; E \Rightarrow y : B.(\exists x : \mathbf{Bool}.Q)}$$

Outline

Verifying Hello Quantum World

```
hqw : {emp} r : Bool {emp ∧ Id(r, false)}
    = do q ⇐ mkQbit false;
         measQbit q
```

At the logic level:

```
hqw : {emp} r : Bool {emp ∧ Id(r, false)}
-- P0: emp
    = do q ⇐ mkQbit false;
-- P1: P0 ∘ (q ↦ |0⟩)
         measQbit q
-- P2: P1 ∘ ((q ↦ -) ⊸ emp)
```

Successful type checking implies correctness
of the program to given specifications.

Verifying Quantum Coin Toss

```
rnd : {emp} r : Bool {emp}
    = do q ⇐ mkQbit false;
         applyU (H q);
         measQbit q
```

At the logic level:

```
rnd : {emp} r : Bool {emp}
-- P0: emp
    = do q ⇐ mkQbit false;
-- P1: P0 ∘ (q ↦ |0⟩)
         applyU (H q);
-- P2: P1 ∘ ((q ↦ |0⟩) ⊸ (q ↦ |+⟩))
         measQbit q
-- P3: P2 ∘ ((q ↦ -) ⊸ emp)
```

Ongoing & Future Work

Tractable semantics for unitary application

- Unitaries as path-sum actions (Amy, *QPL '18*)
- based on Feyman path integrals

Quantum Assertion Logic

Linear Dependent Type Theory: FKS, *LICS '20*

Circuits as Arrows: VAS06, *Math. Struct. Comput. Sci. 16(3)*

Behavioural Types

- Resource Theories: RSSL19 (Draft)
- Heisenberg Representation of QM: RSSL, *QPL '20*

Conclusion

We combined ideas from Hoare Type Theory and Quantum IO Monad to develop Quantum Hoare Type Theory, a dependently typed functional language with support for quantum computation.

This is ongoing work with potential to be a unified framework for programming, specification, and reasoning about quantum programs.

Many exciting challenges ahead!