Tracking Errors through Types in Quantum Programs

Kesha Hietala, Robert Rand, Michael Hicks

Programming Languages for Quantum Computing
Overview

• Errors, particularly those introduced by gate application, will be prevalent in near-term quantum machines, so need to be taken into account when writing programs.

• It might be useful to have a notion of errors at the *programming language level*

• We have designed a simple type system to track errors in quantum programs, implemented in **QWIRE**
QWIRE

• A small but expressive quantum circuit language
QWIRE

- A small but expressive quantum circuit language
- Embedded in the Coq proof assistant
QWIRE

- A small but expressive quantum circuit language
- Embedded in the Coq proof assistant
QWIRE

- A small but expressive quantum circuit language
- Embedded in the Coq proof assistant
- A linear type system for enforcing no-cloning
QWIRE

- A small but expressive quantum circuit language
- Embedded in the Coq proof assistant
- A linear type system for enforcing no-cloning
- A denotational semantics in terms of density matrices
QWIRE

Inductive Circuit (W : WType) : Type :=
| output : Pat W -> Circuit W
| gate : Gate W1 W2 -> Pat W1 ->
    (Pat W2 -> Circuit W) -> Circuit W
| lift : Pat Bit -> (bool -> Circuit W) ->
    Circuit W.
QWIRE

Inductive Circuit (W : WType) : Type :=
| output : Pat W -> Circuit W
| gate : Gate W1 W2 -> Pat W1 ->
   (Pat W2 -> Circuit W) -> Circuit W
| lift : Pat Bit -> (bool -> Circuit W) ->
   Circuit W.
Inductive Circuit (W : WType) : Type :=
| output : Pat W → Circuit W
| gate : Gate W₁ W₂ → Pat W₁ →
    (Pat W₂ → Circuit W) → Circuit W
| lift : Pat Bit → (bool → Circuit W) →
    Circuit W.

---

p : W
QWIRE

Inductive Circuit (W : WType) : Type :=
  | output : Pat W -> Circuit W
  | gate   : Gate W1 W2 -> Pat W1 ->
            (Pat W2 -> Circuit W) -> Circuit W
  | lift   : Pat Bit -> (bool -> Circuit W) ->
            Circuit W.

    ——— p : W
Inductive Circuit \((W : \text{WType}) : \text{Type} :=\)
| output : \(\text{Pat } W \rightarrow \text{Circuit } W\)
| **gate** : \(\text{Gate } W_1 W_2 \rightarrow \text{Pat } W_1 \rightarrow (\text{Pat } W_2 \rightarrow \text{Circuit } W) \rightarrow \text{Circuit } W\)
| lift : \(\text{Pat } \text{Bit} \rightarrow (\text{bool } \rightarrow \text{Circuit } W) \rightarrow \text{Circuit } W.\)
Inductive Circuit (W : WType) : Type :=
| output : Pat W -> Circuit W
| gate : Gate W1 W2 -> Pat W1 ->
   (Pat W2 -> Circuit W) -> Circuit W
| lift : Pat Bit -> (bool -> Circuit W) ->
   Circuit W.
Inductive Circuit (W : WType) : Type :=
  | output : Pat W -> Circuit W
  | gate   : Gate W1 W2 -> Pat W1 ->
          (Pat W2 -> Circuit W) -> Circuit W
  | lift   : Pat Bit -> (bool -> Circuit W) -> Circuit W.

\[ p : W \]

\[ G \]

\[ W1 \]

\[ W2 \]
Inductive Circuit \((W : \text{WType}) : \text{Type} :=\)

- output : Pat W -> Circuit W
- gate : Gate W1 W2 -> Pat W1 -> (Pat W2 -> Circuit W) -> Circuit W
- lift : Pat Bit -> (bool -> Circuit W) -> Circuit W.

\[
\begin{array}{c}
 p: W \\
\end{array}
\]

\[
\begin{array}{c}
p1 \quad W1 \quad G \quad W2 \\
\end{array}
\]
Inductive Circuit (W : WType) : Type :=
| output : Pat W -> Circuit W
| gate   : Gate W1 W2 -> Pat W1 ->
  (Pat W2 -> Circuit W) -> Circuit W
| lift   : Pat Bit -> (bool -> Circuit W) ->
  Circuit W.

\[ p : W \]
Inductive Circuit (W : WType) : Type :=
| output : Pat W -> Circuit W
| gate   : Gate W1 W2 -> Pat W1 ->
                     (Pat W2 -> Circuit W) -> Circuit W
| lift   : Pat Bit -> (bool -> Circuit W) ->
                     Circuit W.

\[ \text{p : W} \]

\[ \text{p1} \quad W1 \quad \text{G} \quad W2 \quad \text{C} \]
Inductive Circuit \((W : \text{WType}) : \text{Type} :=\)

- \text{output} : \text{Pat} W \rightarrow \text{Circuit} W
- \text{gate} : \text{Gate} W1 W2 \rightarrow \text{Pat} W1 \rightarrow \text{Pat} W2 \rightarrow \text{Circuit} W \rightarrow \text{Circuit} W
- \text{lift} : \text{Pat} \text{Bit} \rightarrow (\text{bool} \rightarrow \text{Circuit} W) \rightarrow \text{Circuit} W.
Wire Types

Inductive $\text{WType} := \text{Qubit} \mid \text{Bit} \mid \text{One} \mid W \otimes W$. 
Wire Types

Inductive WType := Qubit | Bit | One | W ⊗ W.

Inductive Gate : WType -> WType -> Set :=
| U       : Unitary W -> Gate W W
| init0   : Gate One Qubit
| new0    : Gate One Bit
| meas    : Gate Qubit Bit
| discard : Gate Bit One.
Wire Types

Inductive \( \text{WType} := \text{Qubit} | \text{Bit} | \text{One} | W \otimes W \).

Inductive \( \text{Gate : WType -> WType -> Set} := \)
| \( U \) : Unitary \( W \rightarrow \text{Gate W W} \)
| \( \text{init0} \) : \( \text{Gate One Qubit} \)
| \( \text{new0} \) : \( \text{Gate One Bit} \)
| \( \text{meas} \) : \( \text{Gate Qubit Bit} \)
| \( \text{discard} \) : \( \text{Gate Bit One} \).
What *Kind* of Qubit?
What *Kind* of Qubit?

- *Physical qubits*: Individual machine qubits, prone to error/degradation.
What *Kind* of Qubit?

- *Physical qubits*: Individual machine qubits, prone to error/degradation.

- *Logical qubits*: 100 - 1000s of physical qubits, assumed to be highly error tolerant and regularly corrected.
What *Kind* of Qubit?

- **Physical qubits**: Individual machine qubits, prone to error/degradation.

- **Logical qubits**: 100 - 1000s of physical qubits, assumed to be highly error tolerant and regularly corrected.

- **Simple Encoded Qubits**: Some small number of physical qubits (say, 9) correspond to a qubit.
What *Kind* of Qubit?

- *Physical qubits*: Individual machine qubits, prone to error/degradation.

- *Logical qubits*: 100 - 1000s of physical qubits, assumed to be highly error tolerant and regularly corrected.

- *Simple Encoded Qubits*: Some small number of physical qubits (say, 9) correspond to a qubit.

\[ |0\rangle_E = (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \]

\[ |1\rangle_E = (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \]
Adding Errors

Inductive $\text{WType} := \text{Qubit} \; k \mid \text{Bit} \mid \text{One} \mid W \otimes W$.

Inductive $\text{Gate} :=$

\[
\begin{array}{c}
| \text{U} | \text{init0} | \text{init1} | \text{meas} | \text{discard} | \text{EC}.
\end{array}
\]
Adding Errors

Inductive WType := Qubit $k$ | Bit | One | $W \otimes W$.

Inductive Gate :=
   | U | init0 | init1 | meas | discard | EC.
Adding Errors

Inductive WType := Qubit \( k \) | Bit | One | \( W \otimes W \).

Inductive Gate :=
  | U | init0 | init1 | meas | discard | EC.
Error Prone Gates

\[
\text{Inductive Gate} : \text{nat} \rightarrow \text{WType} \rightarrow \text{WType} \rightarrow \text{Set} := \\
| \text{U} : (U : \text{Unitary k W}) \rightarrow \\
\quad \text{Gate k W} (\text{map W} (k + \text{sum_err W})) \\
| \text{init0} : \text{Gate} \, \epsilon_i \, \text{One} (\text{Qubit} \, \epsilon_i) \\
| \text{new0} : \text{Gate} \, 0 \, \text{One} \, \text{Bit} \\
| \text{meas} : \text{Gate} \, \epsilon_m \, (\text{Qubit} \, n) \, \text{Bit} \\
| \text{discard} : \text{Gate} \, 0 \, \text{Bit} \, \text{One} \\
| \text{EC} : \text{Gate} \, \epsilon_e \, (\text{Qubit} \, n) \, (\text{Qubit} \, 0).
\]
Error Prone Gates

Inductive Gate : \texttt{nat} -> WType -> WType -> Set :=
| U       : (U : Unitary k W) ->
          Gate k W (map W (k + sum_err W))
| init0   : Gate \epsilon_i One (Qubit \epsilon_i)
| new0    : Gate 0 One Bit
| meas    : Gate \epsilon_m (Qubit n) Bit
| discard : Gate 0 Bit One
| EC      : Gate \epsilon_e (Qubit n) (Qubit 0).

error term
Error Prone Gates

Inductive Gate : \texttt{nat} \to \texttt{WType} \to \texttt{WType} \to \texttt{Set} :=
| \texttt{U} : (U : \texttt{Unitary} \, k \, \texttt{W}) \to
  \texttt{Gate} \, k \, \texttt{W} \, (\texttt{map} \, \texttt{W} \, (k + \texttt{sum\_err} \, \texttt{W}))
| \texttt{init0} : \texttt{Gate} \, \epsilon_i \, \texttt{One} \, (\texttt{Qubit} \, \epsilon_i)
| \texttt{new0} : \texttt{Gate} \, 0 \, \texttt{One} \, \texttt{Bit}
| \texttt{meas} : \texttt{Gate} \, \epsilon_m \, (\texttt{Qubit} \, n) \, \texttt{Bit}
| \texttt{discard} : \texttt{Gate} \, 0 \, \texttt{Bit} \, \texttt{One}
| \texttt{EC} : \texttt{Gate} \, \epsilon_e \, (\texttt{Qubit} \, n) \, (\texttt{Qubit} \, 0).
Error Prone Gates

Inductive Gate : nat -> WType -> WType -> Set :=
| U       : (U : Unitary k W) -> Gate k W (map W (k + sum_err W))
| init0   : Gate One (Qubit ε_i)
| new0    : Gate 0 One Bit
| meas    : Gate ε_m (Qubit n) Bit
| discard : Gate 0 Bit One
| EC      : Gate ε_c (Qubit n) (Qubit 0).
Error Prone Gates

Inductive Gate : nat -> WType -> WType -> Set :=
| U       : (U : Unitary k W) -> Gate k W (map W (k + sum_err W))
| init0   : Gate One (Qubit ε_i)
| new0    : Gate 0 One Bit
| meas    : Gate ε_m (Qubit n) Bit
| discard : Gate 0 Bit One
| EC      : Gate ε_e (Qubit n) (Qubit 0) .
Fault Tolerance
Fault Tolerance

- We say that an operation $O$ is fault tolerant if, provided there are not too many errors on the input, (i) $O$ does not introduce too many errors and (ii) $O$ performs the intended operation.
Fault Tolerance

• We say that an operation $O$ is fault tolerant if, provided there are not too many errors on the input, (i) $O$ does not introduce too many errors and (ii) $O$ performs the intended operation.

• We assume the existence of an *ideal decoder* that corrects up to $t$ errors.
Fault Tolerance

- We say that an operation $O$ is fault tolerant if, provided there are not too many errors on the input, (i) $O$ does not introduce too many errors and (ii) $O$ performs the intended operation.

- We assume the existence of an *ideal decoder* that corrects up to $t$ errors.

- Unitary operation $U$ is fault tolerant if $\sum \epsilon_{in} + \epsilon_{gate} \leq t$ implies...
Fault Tolerance

• We say that an operation $O$ is fault tolerant if, provided there are not too many errors on the input, (i) $O$ does not introduce too many errors and (ii) $O$ performs the intended operation.

• We assume the existence of an ideal decoder that corrects up to $t$ errors.

• Unitary operation $U$ is fault tolerant if $\sum \epsilon_{in} + \epsilon_{gate} \leq t$ implies

1. $\epsilon_{out} \leq \sum \epsilon_{in} + \epsilon_{gate}$

Gottesman (2009): An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation
Fault Tolerance

- We say that an operation $O$ is fault tolerant if, provided there are not too many errors on the input, (i) $O$ does not introduce too many errors and (ii) $O$ performs the intended operation.

- We assume the existence of an *ideal decoder* that corrects up to $t$ errors.

- Unitary operation $U$ is fault tolerant if $\sum e_{in} + e_{gate} \leq t$ implies
  
  1. $e_{out} \leq \sum e_{in} + e_{gate}$
  2. $decode(U; |\psi\rangle) = U_{ideal}; \, decode(|\psi\rangle)$

Gottesman (2009): An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation
Error Correction Gates
Error Correction Gates

• Error correction gate $EC$ is fault tolerant if
Error Correction Gates

- Error correction gate $EC$ is fault tolerant if

1. $c_{out} \leq t$
Error Correction Gates

• Error correction gate $EC$ is fault tolerant if

1. $\epsilon_{out} \leq t$

2. $\epsilon_{in} + \epsilon_{gate} \leq t$ implies

$$decode(EC; |\psi\rangle) = decode(|\psi\rangle)$$
Checking Fault Tolerance
Checking Fault Tolerance

- We check these using QWIRE’s type system
Checking Fault Tolerance

• We check these using QWIRE’s type system

• We benefit from linearity — we cannot introduce an error onto an existing qubit, but only produce a new qubit with some number of errors
Checking Fault Tolerance

- We check these using QWIRE’s type system

- We benefit from linearity — we cannot introduce an error onto an existing qubit, but only produce a new qubit with some number of errors

- However, checking fault tolerance is orthogonal to checking linearity.
Typechecking

Inductive Types_Circuit : OCtx -> nat -> Circuit W -> Set :=
  | types_gate : forall {Γ Γ1 Γ1' w1 w2 w}
      (f : Pat w2 -> Circuit w)
      (k t : nat) {p1 : Pat w1} {g : Gate k w1 w2},
    Γ1 ⊢ p1 : Pat ->
    k + sum_err W1 <= t ->
    (forall Γ2 Γ2' (p2 : Pat w2)
      {pf2 : Γ2' == Γ2 · Γ},
      Γ2 ⊢ p2 : Pat -> Types_Circuit Γ2' t (f p2)) ->
    forall {pf1 : Γ1' == Γ1 · Γ},
    Types_Circuit Γ1' t (gate g p1 f).
Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is "correct" because there is no point at which the potential number of errors in an encoded qubit is above three. In the \texttt{bell00} sub-circuit, both qubits start with zero errors and the \texttt{H} gate introduces up to one error on the first qubit. Then the \texttt{CNOT} gate propagates the error to both qubits and adds another potential error. In the \texttt{alice} sub-circuit, the first \texttt{CNOT} gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the \texttt{bob} sub-circuit, the input wire has up to two errors accumulated, and the \texttt{X} gate potentially adds a third, so error correction must be applied before the final \texttt{Z} gate. In the end, the output has up to one accumulated error.

Syntax, Semantics, and Typing.

We implement error types in the linearly-typed \texttt{Q}wire circuit language [4, 9], which is embedded in the Coq proof assistant [10]. We directly use \texttt{Q}wire's syntax, which allows gate application, sequencing, and dynamic lifting (measurement and return of control to a classical computer). We add to \texttt{Q}wire an error correction gate \texttt{EC}.

With a fixed error correcting code, we interpret \texttt{Q}wire's \texttt{Qubit} type as an encoded qubit and assume that all gates are fault-tolerant. We associate with every gate a natural number \( n \) corresponding to the maximum number of errors that it can introduce.

To define fault tolerance we follow the presentation by Gottesman [7, Section 4.2]. First, fix a threshold \( t \), which is the maximum number of errors that the code can correct. Then assume an ideal decoder, which takes an encoded state, corrects any errors, and returns an unencoded state (though not necessarily the "correct" state if too many errors have occurred). The basic requirements for a fault-tolerant operation are that it does not propagate too many errors and that it performs the desired ideal operation. In particular:

• A measurement operation is fault tolerant if it is equivalent to ideal decoding followed by ideal measurement, provided that the total number of errors in the incoming state and measurement operation is at most \( t \).

• A state preparation operation is fault tolerant if (i) the operation introduces at most \( t \) errors and (ii) state preparation followed by ideal decoding is equivalent to ideal state preparation.

• A unitary operation is fault tolerant if (i) the number of errors on every output after the operation is at most the sum of the errors introduced by the operation and the errors on all inputs and this quantity is at most \( t \) and (ii) unitary application followed by ideal decoding is equivalent to ideal decoding followed by the ideal unitary operation.

• An error correction operation is fault tolerant if (i) the output of error correction has at most \( t \) errors
Typechecking

Definition \textit{bell} : \text{Box One (Qubit 2 \otimes Qubit 2)} := ...
Typechecking

Definition \texttt{bell} : \texttt{Box One (Qubit 2 \otimes Qubit 2)} := ... 

Definition \texttt{alice} : \texttt{Box (Qubit 0 \otimes Qubit 2) (Bit \otimes Bit)} := ...

Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is "correct" because there is no point at which the potential number of errors in an encoded qubit is above three. In the \texttt{bell00} sub-circuit, both qubits start with zero errors and the \texttt{H} gate introduces up to one error on the first qubit. Then the \texttt{CNOT} gate propagates the error to both qubits and adds another potential error. In the \texttt{alice} sub-circuit, the first \texttt{CNOT} gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the \texttt{bob} sub-circuit, the input wire has up to two errors accumulated, and the \texttt{X} gate potentially adds a third, so error correction must be applied before the final \texttt{Z} gate. In the end, the output has up to one accumulated error.

Syntax, Semantics, and Typing.

We implement error types in the linearly-typed \texttt{Q wire} circuit language \cite{4,9}, which is embedded in the Coq proof assistant \cite{10}. We directly use \texttt{Q wire}'s syntax, which allows gate application, sequencing, and dynamic lifting (measurement and return of control to a classical computer). We add to \texttt{Q wire} an error correction gate \texttt{EC}.

With a fixed error correcting code, we interpret \texttt{Q wire}'s \texttt{Qubit} type as an encoded qubit and assume that all gates are fault-tolerant. We associate with every gate a natural number \( n \) corresponding to the maximum number of errors that it can introduce.

To define fault tolerance we follow the presentation by Gottesman \cite{7,Section 4.2}. First, fix a threshold \( t \), which is the maximum number of errors that the code can correct. Then assume an ideal decoder, which takes an encoded state, corrects any errors, and returns an unencoded state (though not necessarily the "correct" state if too many errors have occurred). The basic requirements for a fault-tolerant operation are that it does not propagate too many errors and that it performs the desired ideal operation. In particular:

\begin{itemize}
  \item A measurement operation is fault tolerant if it is equivalent to ideal decoding followed by ideal measurement, provided that the total number of errors in the incoming state and measurement operation is at most \( t \).
  \item A state preparation operation is fault tolerant if (i) the operation introduces at most \( t \) errors and (ii) state preparation followed by ideal decoding is equivalent to ideal state preparation.
  \item A unitary operation is fault tolerant if (i) the number of errors on every output after the operation is at most the sum of the errors introduced by the operation and the errors on all inputs and this quantity is at most \( t \) and (ii) unitary application followed by ideal decoding is equivalent to ideal decoding followed by the ideal unitary operation.
  \item An error correction operation is fault tolerant if (i) the output of error correction has at most \( t \) errors.
\end{itemize}
Typechecking

Definition bell : Box One (Qubit 2 ⊗ Qubit 2) := ...

Definition alice : Box (Qubit 0 ⊗ Qubit 2) (Bit ⊗ Bit) := ...

Definition bob : Box (Bit ⊗ Bit ⊗ Qubit 2) (Qubit 1) := ...
Typechecking

**Definition** bell : Box One (Qubit 2 ⊗ Qubit 2) := ...

**Definition** alice : Box (Qubit 0 ⊗ Qubit 2) (Bit ⊗ Bit) := ...

**Definition** bob : Box (Bit ⊗ Bit ⊗ Qubit 2) (Qubit 1) := ...

**Definition** teleport : Box (Qubit 0) (Qubit 1) :=

---

Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is "correct" because there is no point at which the potential number of errors in an encoded qubit is above three. In the bell00 sub-circuit, both qubits start with zero errors and the $H$ gate introduces up to one error on the first qubit. Then the CNOT gate propagates the error to both qubits and adds another potential error. In the alice sub-circuit, the first CNOT gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the bob sub-circuit, the input wire has up to two errors accumulated, and the $X$ gate potentially adds a third, so error correction must be applied before the final $Z$ gate. In the end, the output has up to one accumulated error.

Syntax, Semantics, and Typing.
We implement error types in the linearly-typed $Q$ wire circuit language [4, 9], which is embedded in the Coq proof assistant [10]. We directly use $Q$ wire's syntax, which allows gate application, sequencing, and dynamic lifting (measurement and return of control to a classical computer). We add to $Q$ wire an error correction gate $EC$. With a fixed error correcting code, we interpret $Q$ wire's Qubit type as an encoded qubit and assume that all gates are fault-tolerant. We associate with every gate a natural number $n$ corresponding to the maximum number of errors that it can introduce.

To define fault tolerance we follow the presentation by Gottesman [7, Section 4.2]. First, fix a threshold $t$, which is the maximum number of errors that the code can correct. Then assume an ideal decoder, which takes an encoded state, corrects any errors, and returns an unencoded state (though not necessarily the "correct" state if too many errors have occurred). The basic requirements for a fault-tolerant operation are that it does not propagate too many errors and that it performs the desired ideal operation. In particular:

- A measurement operation is fault tolerant if it is equivalent to ideal decoding followed by ideal measurement, provided that the total number of errors in the incoming state and measurement operation is at most $t$.
- A state preparation operation is fault tolerant if (i) the operation introduces at most $t$ errors and (ii) state preparation followed by ideal decoding is equivalent to ideal state preparation.
- A unitary operation is fault tolerant if (i) the number of errors on every output after the operation is at most the sum of the errors introduced by the operation and the errors on all inputs and this quantity is at most $t$ and (ii) unitary application followed by ideal decoding is equivalent to ideal decoding followed by the ideal unitary operation.
- An error correction operation is fault tolerant if (i) the output of error correction has at most $t$ errors.
Typechecking

Definition bell : Box One (Qubit 2 ⊗ Qubit 2) := ...

Definition alice : Box (Qubit 0 ⊗ Qubit 2) (Bit ⊗ Bit) := ...

Definition bob : Box (Bit ⊗ Bit ⊗ Qubit 2) (Qubit 1) := ...

Definition teleport : Box (Qubit 0) (Qubit 1) :=

Lemma teleport_EC_WT : Typed_Box teleport 3.
Proof. type_check. Qed.
Type Inference

Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is “correct” because there is no point at which the potential number of errors in an encoded qubit is above three. In the `bell00` sub-circuit, both qubits start with zero errors and the `H` gate introduces up to one error on the first qubit. Then the `CNOT` gate propagates the error to both qubits and adds another potential error. In the `alice` sub-circuit, the first `CNOT` gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the `bob` sub-circuit, the input wire has up to two errors accumulated, and the `X` gate potentially adds a third, so error correction must be applied before the final `Z` gate. In the end, the output has up to one accumulated error.

Syntax, Semantics, and Typing.

We implement error types in the linearly-typed `Qwire` circuit language \[4, 9\], which is embedded in the Coq proof assistant \[10\]. We directly use `Qwire`'s syntax, which allows gate application, sequencing, and dynamic lifting (measurement and return of control to a classical computer). We add to `Qwire` an error correction gate `EC`.

With a fixed error correcting code, we interpret `Qwire`'s `Qubit` type as an encoded qubit and assume that all gates are fault-tolerant. We associate with every gate a natural number \( n \) corresponding to the maximum number of errors that it can introduce.

To define fault tolerance we follow the presentation by Gottesman \[7, Section 4.2\]. First, fix a threshold \( t \), which is the maximum number of errors that the code can correct. Then assume an ideal decoder, which takes an encoded state, corrects any errors, and returns an unencoded state (though not necessarily the “correct” state if too many errors have occurred). The basic requirements for a fault-tolerant operation are that it does not propagate too many errors and that it performs the desired ideal operation. In particular:

- A measurement operation is fault tolerant if it is equivalent to ideal decoding followed by ideal measurement, provided that the total number of errors in the incoming state and measurement operation is at most \( t \).
- A state preparation operation is fault tolerant if (i) the operation introduces at most \( t \) errors and (ii) state preparation followed by ideal decoding is equivalent to ideal state preparation.
- A unitary operation is fault tolerant if (i) the number of errors on every output after the operation is at most the sum of the errors introduced by the operation and the errors on all inputs and this quantity is at most \( t \) and (ii) unitary application followed by ideal decoding is equivalent to ideal decoding followed by the ideal unitary operation.
- An error correction operation is fault tolerant if (i) the output of error correction has at most \( t \) errors.
Type Inference

Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is “correct” because there is no point at which the potential number of errors in an encoded qubit is above three. In the bell00 sub-circuit, both qubits start with zero errors and the H gate introduces up to one error on the first qubit. Then the CNOT gate propagates the error to both qubits and adds another potential error. In the alice sub-circuit, the first CNOT gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the bob sub-circuit, the input wire has up to two errors accumulated, and the X gate potentially adds a third, so error correction must be applied before the final Z gate. In the end, the output has up to one accumulated error.

Syntax, Semantics, and Typing.

We implement error types in the linearly-typed Qwire circuit language [4, 9], which is embedded in the Coq proof assistant [10]. We directly use Qwire’s syntax, which allows gate application, sequencing, and dynamic lifting (measurement and return of control to a classical computer). We add to Qwire an error correction gate EC.

With a fixed error correcting code, we interpret Qwire’s Qubit type as an encoded qubit and assume that all gates are fault-tolerant. We associate with every gate a natural number n corresponding to the maximum number of errors that it can introduce.

To define fault tolerance we follow the presentation by Gottesman [7, Section 4.2]. First, fix a threshold t, which is the maximum number of errors that the code can correct. Then assume an ideal decoder, which takes an encoded state, corrects any errors, and returns an unencoded state (though not necessarily the “correct” state if too many errors have occurred). The basic requirements for a fault-tolerant operation are that it does not propagate too many errors and that it performs the desired ideal operation. In particular:

- A measurement operation is fault tolerant if it is equivalent to ideal decoding followed by ideal measurement, provided that the total number of errors in the incoming state and measurement operation is at most $t$.
- A state preparation operation is fault tolerant if (i) the operation introduces at most $t$ errors and (ii) state preparation followed by ideal decoding is equivalent to ideal state preparation.
- A unitary operation is fault tolerant if (i) the number of errors on every output after the operation is at most the sum of the errors introduced by the operation and the errors on all inputs and this quantity is at most $t$ and (ii) unitary application followed by ideal decoding is equivalent to ideal decoding followed by the ideal unitary operation.
- An error correction operation is fault tolerant if (i) the output of error correction has at most $t$ errors and

\[\textbf{Definition} \ \text{bell} \ := \ \ldots\]
Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is “correct” because there is no point at which the potential number of errors in an encoded qubit is above three. In the \texttt{bell00} sub-circuit, both qubits start with zero errors and the \texttt{H} gate introduces up to one error on the first qubit. Then the \texttt{CNOT} gate propagates the error to both qubits and adds another potential error. In the \texttt{alice} sub-circuit, the first \texttt{CNOT} gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the \texttt{bob} sub-circuit, the input wire has up to two errors accumulated, and the \texttt{X} gate potentially adds a third, so error correction must be applied before the final \texttt{Z} gate. In the end, the output has up to one accumulated error.

We implement error types in the linearly-typed \texttt{Q} wire circuit language \cite{4,9}, which is embedded in the Coq proof assistant \cite{10}. We directly use \texttt{Q} wire’s syntax, which allows gate application, sequencing, and dynamic lifting (measurement and return of control to a classical computer). We add to \texttt{Q} wire an error correction gate \texttt{EC}.

With a fixed error correcting code, we interpret \texttt{Q} wire’s \texttt{Qubit} type as an encoded qubit and assume that all gates are fault-tolerant. We associate with every gate a natural number \( n \) corresponding to the maximum number of errors that it can introduce.

To define fault tolerance we follow the presentation by Gottesman \cite[Section 4.2]{7}. First, fix a threshold \( t \), which is the maximum number of errors that the code can correct. Then assume an ideal decoder, which takes an encoded state, corrects any errors, and returns an unencoded state (though not necessarily the “correct” state if too many errors have occurred). The basic requirements for a fault-tolerant operation are that it does not propagate too many errors and that it performs the desired ideal operation. In particular:

1. A measurement operation is fault tolerant if it is equivalent to ideal decoding followed by ideal measurement, provided that the total number of errors in the incoming state and measurement operation is at most \( t \).
2. A state preparation operation is fault tolerant if (i) the operation introduces at most \( t \) errors and (ii) state preparation followed by ideal decoding is equivalent to ideal state preparation.
3. A unitary operation is fault tolerant if (i) the number of errors on every output after the operation is at most the sum of the errors introduced by the operation and the errors on all inputs and this quantity is at most \( t \) and (ii) unitary application followed by ideal decoding is equivalent to ideal decoding followed by the ideal unitary operation.
4. An error correction operation is fault tolerant if (i) the output of error correction has at most \( t \) errors.
**Definition** bell := ...
Type Inference

**Definition** bell := ...

**Definition** alice : _ (_ 0 ⊗ _ 2) _ := ...

Check

Box One (Qubit 2 ⊗ Qubit 2)

\[
\begin{align*}
|\psi\rangle & \quad EC \quad H \quad \text{alice} \\
|0\rangle & \quad H \\
|0\rangle & \quad \text{bell00} \\
|\psi\rangle & \quad X \quad EC \quad Z \quad \text{bob}
\end{align*}
\]
Type Inference

Definition bell := ...

Definition alice : _ (_ 0 ⊗ _ 2) _ := ...

Check

: Box One (Qubit 2 ⊗ Qubit 2)

: Box (Qubit 0 ⊗ Qubit 2) (Bit ⊗ Bit)

Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is "correct" because there is no point at which the potential number of errors in an encoded qubit is above three. In the bell00 sub-circuit, both qubits start with zero errors and the $H$ gate introduces up to one error on the first qubit. Then the $CNOT$ gate propagates the error to both qubits and adds another potential error. In the alice sub-circuit, the first $CNOT$ gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the bob sub-circuit, the input wire has up to two errors accumulated, and the $X$ gate potentially adds a third, so error correction must be applied before the final $Z$ gate. In the end, the output has up to one accumulated error.
Type Inference

Definition bell := …

Definition alice : _ (_ 0 ⊗ _ 2) _ := …

Definition bob : _ (_ ⊗ _ ⊗ _ 2) _ := …

Check: Box One (Qubit 2 ⊗ Qubit 2)

Definition alice : _ (_ 0 ⊗ _ 2) _ := … : Box (Qubit 0 ⊗ Qubit 2) (Bit ⊗ Bit)

Definition bob : _ (_ ⊗ _ ⊗ _ 2) _ := …
Type Inference

\[ |\psi\rangle \]

**Definition** bell := ...

: Box One (Qubit 2 \(\otimes\) Qubit 2)

**Definition** alice : _ (_ 0 \(\otimes\) _ 2) _ := ...

: Box (Qubit 0 \(\otimes\) Qubit 2) (Bit \(\otimes\) Bit)

**Definition** bob : _ (_ \(\otimes\) _ \(\otimes\) _ 2) _ := ...

: Box (Bit \(\otimes\) Bit \(\otimes\) Qubit 2) (Qubit 1)

Check

Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is "correct" because there is no point at which the potential number of errors in an encoded qubit is above three. In the \(bell00\) sub-circuit, both qubits start with zero errors and the \(H\) gate introduces up to one error on the first qubit. Then the \(CNOT\) gate propagates the error to both qubits and adds another potential error. In the \(alice\) sub-circuit, the first \(CNOT\) gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the \(bob\) sub-circuit, the input wire has up to two errors accumulated, and the \(X\) gate potentially adds a third, so error correction must be applied before the final \(Z\) gate. In the end, the output has up to one accumulated error.

Syntax, Semantics, and Typing.

We implement error types in the linearly-typed \(Q\) wire circuit language [4, 9], which is embedded in the Coq proof assistant [10]. We directly use \(Q\) wire's syntax, which allows gate application, sequencing, and dynamic lifting (measurement and return of control to a classical computer). We add to \(Q\) wire an error correction gate \(EC\).

With a fixed error correcting code, we interpret \(Q\) wire's Qubit type as an encoded qubit and assume that all gates are fault-tolerant. We associate with every gate a natural number \(n\) corresponding to the maximum number of errors that it can introduce.

To define fault tolerance we follow the presentation by Gottesman [7, Section 4.2]. First, fix a threshold \(t\), which is the maximum number of errors that the code can correct. Then assume an ideal decoder, which takes an encoded state, corrects any errors, and returns an unencoded state (though not necessarily the "correct" state if too many errors have occurred). The basic requirements for a fault-tolerant operation are that it does not propagate too many errors and that it performs the desired ideal operation. In particular:

- A measurement operation is fault tolerant if it is equivalent to ideal decoding followed by ideal measurement, provided that the total number of errors in the incoming state and measurement operation is at most \(t\).
- A state preparation operation is fault tolerant if (i) the operation introduces at most \(t\) errors and (ii) state preparation followed by ideal decoding is equivalent to ideal state preparation.
- A unitary operation is fault tolerant if (i) the number of errors on every output after the operation is at most the sum of the errors introduced by the operation and the errors on all inputs and this quantity is at most \(t\) and (ii) unitary application followed by ideal decoding is equivalent to ideal decoding followed by the ideal unitary operation.
- An error correction operation is fault tolerant if (i) the output of error correction has at most \(t\) errors

2
Type Inference

Definition bell := ...

Definition alice : _ (_ 0 ⊗ _ 2) _ := ...

Definition bob : _ (_ ⊗ _ ⊗ 2) _ := ...

Definition teleport := ...

Check

: Box One (Qubit 2 ⊗ Qubit 2)

: Box (Qubit 0 ⊗ Qubit 2) (Bit ⊗ Bit)

: Box (Bit ⊗ Bit ⊗ Qubit 2) (Qubit 1)
Consider the circuit in Figure 1 (the corresponding program is shown in Figure 2). Assume our error-correcting code can correct up to three errors and that state preparation, measurement, and error correction are perfect, while unitary application may introduce up to one error. Our goal is to determine whether the error correction operations that we have added will successfully counteract potential errors. To do this, we track how errors propagate through the circuit and ensure that the number of accumulated errors in each encoded qubit does not exceed the threshold.

We can see that the circuit above is “correct” because there is no point at which the potential number of errors in an encoded qubit is above three. In the \texttt{bell00} sub-circuit, both qubits start with zero errors and the \texttt{H} gate introduces up to one error on the first qubit. Then the \texttt{CNOT} gate propagates the error to both qubits and adds another potential error. In the \texttt{alice} sub-circuit, the first \texttt{CNOT} gate now emits up to three errors on both qubits, so error correction must be applied to the first qubit before another gate can be safely applied. After the qubits are measured, their error disappears (error is not tracked for the classical bit type, which we assume is stored on classical hardware). In the \texttt{bob} sub-circuit, the input wire has up to two errors accumulated, and the \texttt{X} gate potentially adds a third, so error correction must be applied before the final \texttt{Z} gate. In the end, the output has up to one accumulated error.

Syntax, Semantics, and Typing.

We implement error types in the linearly-typed \texttt{Q} wire circuit language \cite{4, 9}, which is embedded in the Coq proof assistant \cite{10}. We directly use \texttt{Q} wire's syntax, which allows gate application, sequencing, and dynamic lifting (measurement and return of control to a classical computer). We add to \texttt{Q} wire an error correction gate \texttt{EC}.

With a fixed error correcting code, we interpret \texttt{Q} wire's \texttt{Qubit} type as an encoded qubit and assume that all gates are fault-tolerant. We associate with every gate a natural number \(n\) corresponding to the maximum number of errors that it can introduce.

To define fault tolerance we follow the presentation by Gottesman \cite{7, Section 4.2}. First, fix a threshold \(t\), which is the maximum number of errors that the code can correct. Then assume an ideal decoder, which takes an encoded state, corrects any errors, and returns an unencoded state (though not necessarily the “correct” state if too many errors have occurred). The basic requirements for a fault-tolerant operation are that it does not propagate too many errors and that it performs the desired ideal operation. In particular:

- A measurement operation is fault tolerant if it is equivalent to ideal decoding followed by ideal measurement, provided that the total number of errors in the incoming state and measurement operation is at most \(t\).
- A state preparation operation is fault tolerant if (i) the operation introduces at most \(t\) errors and (ii) state preparation followed by ideal decoding is equivalent to ideal state preparation.
- A unitary operation is fault tolerant if (i) the number of errors on every output after the operation is at most the sum of the errors introduced by the operation and the errors on all inputs and this quantity is at most \(t\) and (ii) unitary application followed by ideal decoding is equivalent to ideal decoding followed by the ideal unitary operation.
- An error correction operation is fault tolerant if (i) the output of error correction has at most \(t\) errors.
Questions

- How can we reflect our error handling back into QWIRE’s denotational semantics?

- What about errors due to decoherence, that aren’t captured by gate application?

- What are better ways of handling errors in a type system or broader reasoning system?