$\mathcal{Q}$WIRE Practice: Formal Verification of Quantum Circuits in Coq

Robert Rand, Jennifer Paykin, Steve Zdancewic

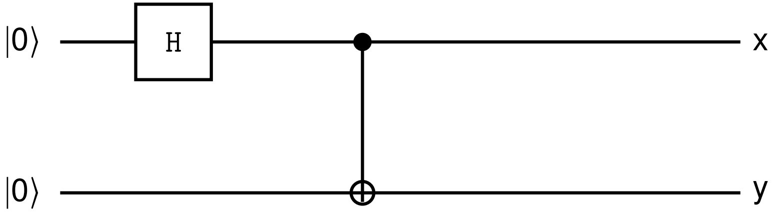
University of Pennsylvania

Quantum Physics and Logic, 2017

$\mathcal{Q}$WIRE

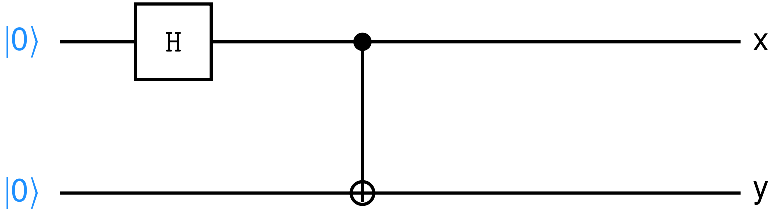
- A high-level language for generating quantum circuits
  - following Quipper and LIQ$Ui|\rangle$
- A minimal set of primitives for circuit construction
- Programs are guaranteed to correspond to realizable quantum computations
  - using linear types, as in the Quantum Lambda Calculus

Preparing a Bell state



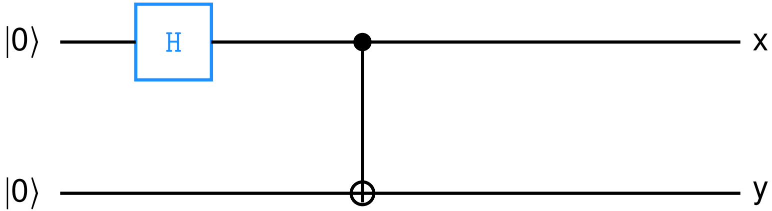

```
Definition bell00 : Box One (Qubit ⊗ Qubit).
  box () ⇒
    gate x     ← init0 @();
    gate y     ← init0 @();
    gate x     ← H @x;
    gate (x,y) ← CNOT @(x,y);
    output (x,y)
Defined.
```

Preparing a Bell state



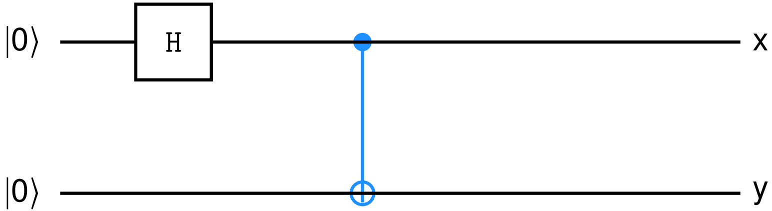

```
Definition bell00 : Box One (Qubit ⊗ Qubit).
  box () ⇒
    gate x      ← init0 @();
    gate y      ← init0 @();
    gate x      ← H @x;
    gate (x,y) ← CNOT @(x,y);
    output (x,y)
Defined.
```

Preparing a Bell state



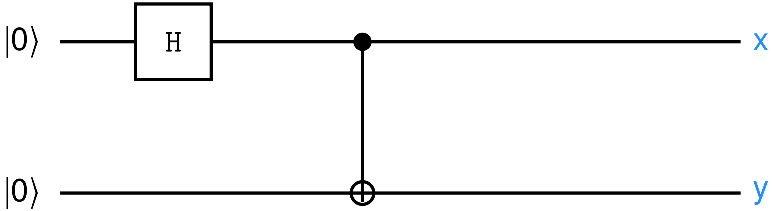

```
Definition bell00 : Box One (Qubit ⊗ Qubit).
  box () ⇒
    gate x     ← init0 @();
    gate y     ← init0 @();
    gate x     ← H @x;
    gate (x,y) ← CNOT @(x,y);
    output (x,y)
Defined.
```

Preparing a Bell state



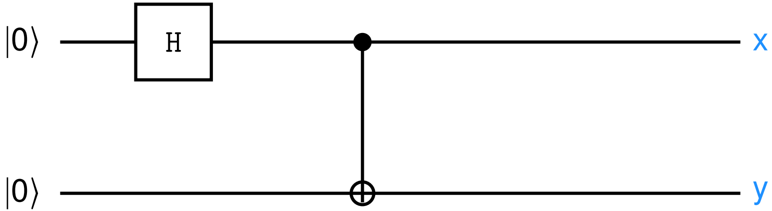

```
Definition bell00 : Box One (Qubit ⊗ Qubit).
  box () ⇒
    gate x     ← init0 @();
    gate y     ← init0 @();
    gate x     ← H @x;
    gate (x,y) ← CNOT @(x,y);
    output (x,y)
Defined.
```

Preparing a Bell state



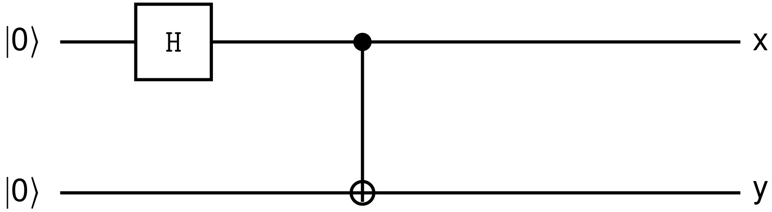

```
Definition bell00 : Box One (Qubit ⊗ Qubit).
  box () ⇒
    gate x     ← init0 @();
    gate y     ← init0 @();
    gate x     ← H @x;
    gate (x,y) ← CNOT @(x,y);
    output (x,y)
Defined.
```

Preparing a Bell state



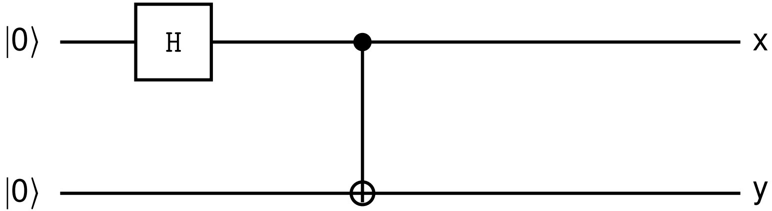

```
Definition bell00 : Box One (Qubit ⊗ Qubit).
  box () ⇒
    gate x     ← init0 @();
    gate y     ← init0 @();
    gate x     ← H @x;
    gate (x,y) ← CNOT @(x,y);
    output (x,y)
Defined.
```

Preparing a Bell state



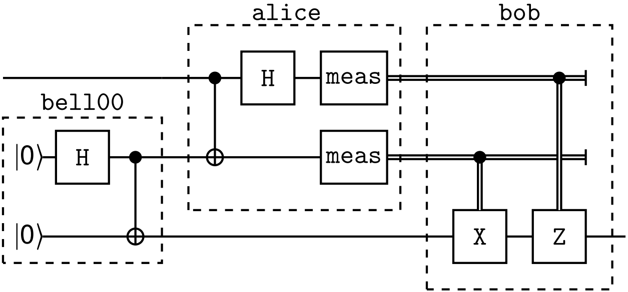

```
Definition bell00 : Box One (Qubit ⊗ Qubit).
  box () ⇒
    gate x     ← init0 @();
    gate y     ← init0 @();
    gate x     ← H @x;
    gate (x,y) ← CNOT @(x,y);
    output (x,y)
Defined.
```

Preparing a Bell state



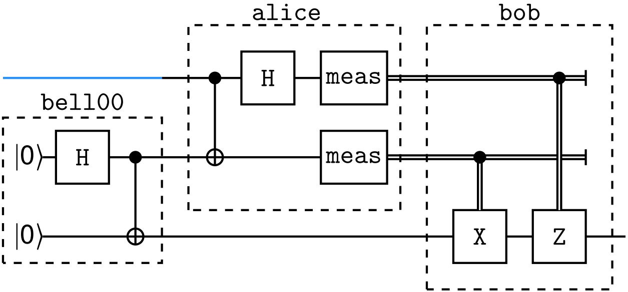

```
Definition bell00 : Box One (Qubit ⊗ Qubit).
  box () ⇒
    gate x     ← init0 @();
    gate y     ← init0 @();
    gate x     ← H @x;
    gate (x,y) ← CNOT @(x,y);
    output (x,y)
Defined.
```

Quantum Teleportation



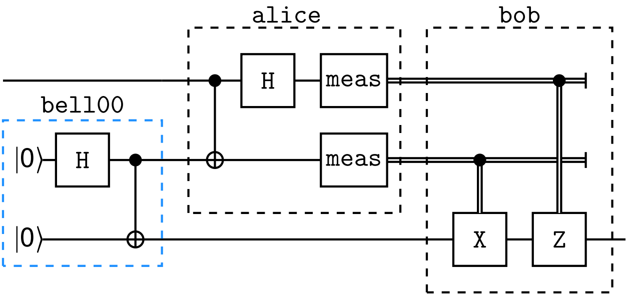

```
Definition teleport : Box Qubit Qubit.
  box t ⇒
    let_ (p,q) ← unbox bell00 ();
    let_ (a,b) ← unbox alice (t,p);
    let_ t'    ← unbox bob (a,b,q);
    output t'
Defined.
```

Quantum Teleportation



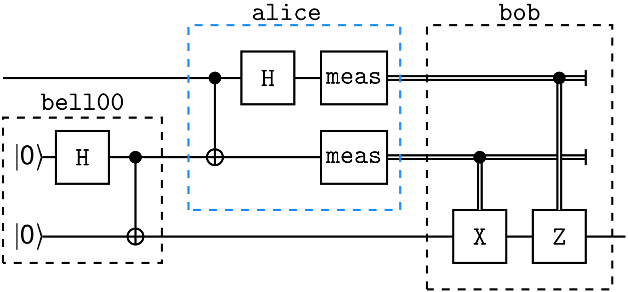

```
Definition teleport : Box Qubit Qubit.
  box t ⇒
    let_ (p,q) ← unbox bell00 ();
    let_ (a,b) ← unbox alice (t,p);
    let_ t'    ← unbox bob (a,b,q);
    output t'
Defined.
```

Quantum Teleportation



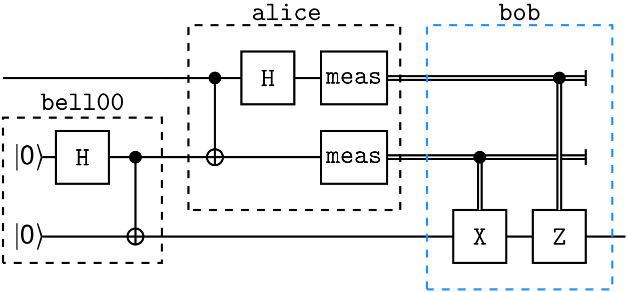

```
Definition teleport : Box Qubit Qubit.
  box t ⇒
    let_ (p,q) ← unbox bell00 ();
    let_ (a,b) ← unbox alice (t,p);
    let_ t'    ← unbox bob (a,b,q);
    output t'
Defined.
```

Quantum Teleportation



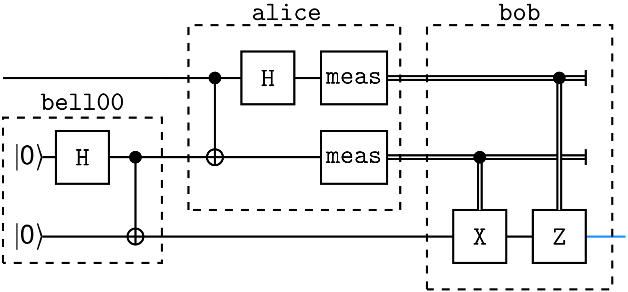

```
Definition teleport : Box Qubit Qubit.
  box t ⇒
    let_ (p,q) ← unbox bell00 ();
    let_ (a,b) ← unbox alice (t,p);
    let_ t'    ← unbox bob (a,b,q);
    output t'
Defined.
```

Quantum Teleportation




```
Definition teleport : Box Qubit Qubit.
  box t ⇒
      let_ (p,q) ← unbox bell00 ();
      let_ (a,b) ← unbox alice (t,p);
      let_ t'    ← unbox bob (a,b,q);
      output t'
Defined.
```

Quantum Teleportation




```
Definition teleport : Box Qubit Qubit.
  box t ⇒
     let_ (p,q) ← unbox bell00 ();
     let_ (a,b) ← unbox alice (t,p);
     let_ t'    ← unbox bob (a,b,q);
     output t'
Defined.
```

Non-Quantum Circuits

```
Definition clone : Box Qubit (Qubit ⊗ Qubit).
  box q ⇒ output (q,q).
Abort.
```

Non-Quantum Circuits

```
Definition [illegible]on[illegible] Box Qubit (Qubit ⊗ Qubit).
  box q ⇒ o[illegible] (q,q).
Abort.
```

Non-Quantum Circuits

```
Definition clone : Box Qubit (Qubit ⊗ Qubit).
  box q ⇒ output (q,q).
Abort.
```

```
Definition delete : Box Qubit One.
  box q ⇒
   gate q ← H @q ;
   output ().
Abort.
```

Non-Quantum Circuits

```
Definition [illegible] Box Qubit (Qubit ⊗ Qubit).
  box q ⇒ o[illegible] (q,q).
Abort.
```

```
Definition [illegible]delete[illegible] Box Qubit One.
  box q ⇒[illegible]
   gate q ← [illegible] ;
   output [illegible]
Abort.
```

Non-Quantum Circuits

```
Definition [illegible]on[illegible] Box Qubit (Qubit ⊗ Qubit).
  box q ⇒ o[illegible] (q,q).
Abort.
```

```
Definition delete[illegible] : Box Qubit One.
  box q ⇒
   gate q ← [illegible] ;
   output [illegible]
Abort.
```

```
Definition recall : Box Qubit Qubit.
  box q ⇒
    gate p ← Z @q ;
    output q.
Abort.
```

Non-Quantum Circuits

```
Definition [illegible]on[illegible] Box Qubit (Qubit ⊗ Qubit).
  box q ⇒ o[illegible] (q,q).
Abort.
```

```
Definition [illegible]delete[illegible]: Box Qubit One.
  box q ⇒ [illegible]
   gate q ← [illegible] ;
   output [illegible]
Abort.
```

```
Definition [illegible]recall[illegible]: Box Qubit Qubit.
  box q ⇒ [illegible]
    gate p ← [illegible]q ;
    output[illegible]
Abort.
```

Linear Types are not Enough

Sources of Errors

- Provide the wrong argument to a phase shift gate.
- Apply a unitary to the wrong wire, or vice-versa.
- Forget to uncompute or measure-discard certain qubits
- Classically compute the wrong circuit.

Approaches

- Simulation
- Debugging
- Unit Testing
- Math + Concentration

Approaches

- Simulation X
- Debugging
- Unit Testing
- Math + Concentration

Approaches

- Simulation X
- Debugging XX
- Unit Testing
- Math + Concentration

Approaches

- Simulation X
- Debugging XX
- Unit Testing $$$
- Math + Concentration

Approaches

- Simulation X
- Debugging XX
- Unit Testing $$$
- Math + Concentration ?

The Coq Proof Assistant

A programming language

An interactive theorem prover

The Coq Proof Assistant

Accomplishments

Mathematics	*Programming*
Feit-Thompson Theorem	CompCert Compiler
Four Color Theorem	CertiKOS

Coq + $\mathcal{Q}$WIRE

- A circuit description language
- A density matrix library
- A `denote` function from circuits to superoperators
- Integrated proofs of program correctness

Denoting Programs

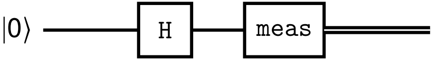
```
Definition denote_unitary : Unitary → Matrix.

Definition denote_gate    : Gate → Superoperator.

Definition denote_circuit : Circuit → Superoperator.
```

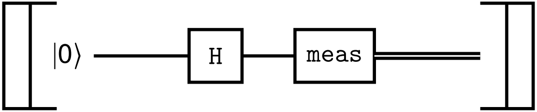
```
Class Denote source target := {denote : source → target}.
Notation "⟦ s ⟧" := (denote s) (at level 10).
```

Denoting a Coin Flip



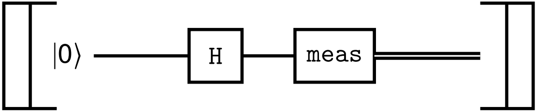

```
Definition coin_flip : Box One Bit.
  box () ⇒
    gate q  ← init0 @();
    gate q  ← H @q;
    gate b  ← meas @q;
    output b.
Defined.
```

Denoting a Coin Flip




```
Lemma fair_toss : ⟦coin_flip⟧ [1]  = [ [1/2,  0]
                                       [0,  1/2] ].
```

Denoting a Coin Flip



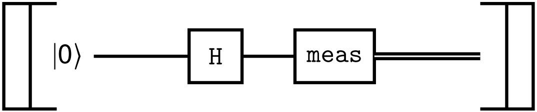
$$=$$

$$|0\rangle\langle 0|\ \mathrm{H}\ |0\rangle\langle 0|\ \mathrm{H}^{\dagger}\ |0\rangle\langle 0|$$

$$+$$

$$|1\rangle\langle 1|\ \mathrm{H}\ |0\rangle\langle 0|\ \mathrm{H}^{\dagger}\ |1\rangle\langle 1|$$

Denoting a Coin Flip

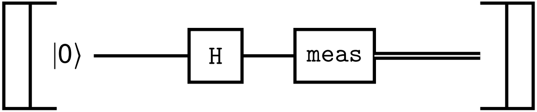


$$= \begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}$$

$$+$$

$$\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}$$

Denoting a Coin Flip



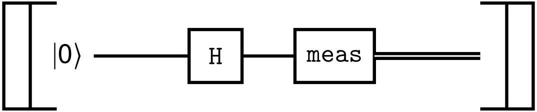
$$=$$

$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix}$$

$$+$$

$$\begin{pmatrix} 0 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

Denoting a Coin Flip



$$=$$

$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

Verified

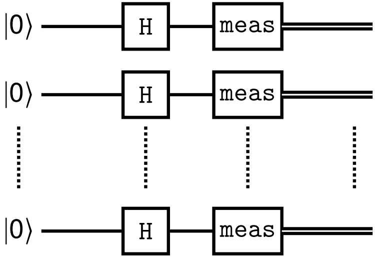
```
Definition fair_coin : Matrix 2 2 :=
  λ x y ⇒ match x, y with
          | 0, 0 ⇒ 1/2
          | 1, 1 ⇒ 1/2
          | _, _ ⇒ 0
          end.

Lemma fair_toss : ⟦ coin_flip⟧  I₁  = fair_coin.
Proof.
  repeat (unfold compose_super, super, swap_list,
          swap_two, pad, apply_new₀, apply_U,
          apply_meas, denote_pat_in; simpl).
  Msimpl.
  prep_matrix_equality.
  unfold even_toss, ket₀, ket₁, Mplus, Mmult, conj_transpose.
  Csimpl.
  destruct x, y; Csimpl; destruct_Csolve. Csolve.
Qed.
```

U:--- **Denotation.v** 86% L681 Git:NoDependencies (Coq Script(0-) +4 company-coq yas hs company Holes Outl)

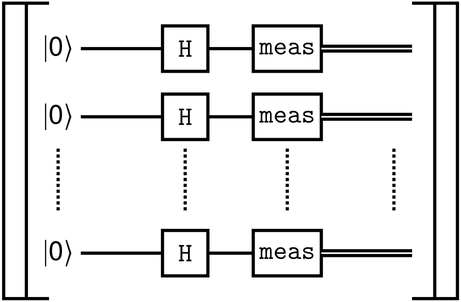
fair_toss is defined

Denoting Coin Flips



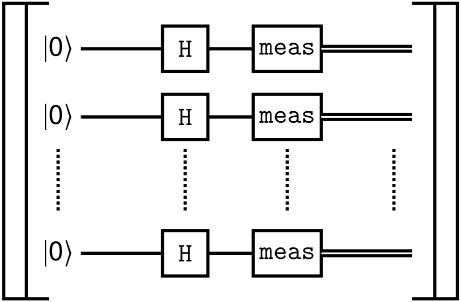

```
(* Generates a binary number between 0 and 2^n *)
Definition uniform (n : ℕ) : Box (n ⊗ One) (n ⊗ Bit) :=
  parallel_copy n coin_flip
```

Denoting Coin Flips



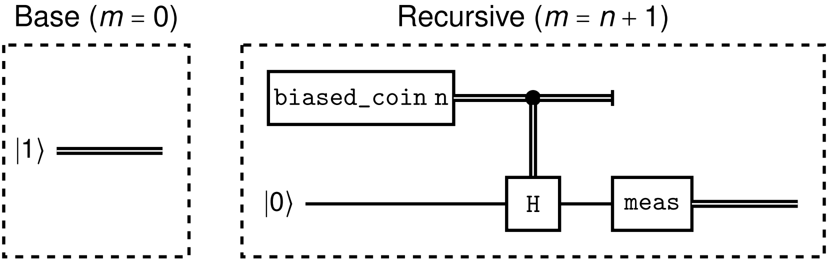
$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

Denoting Coin Flips



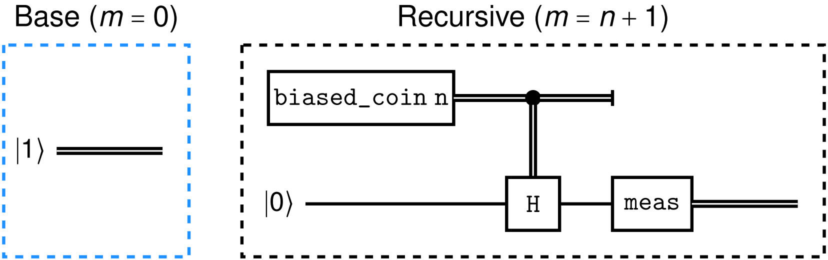
$$\begin{pmatrix} 2^{-n} & 0 & \dots & 0 \\ 0 & 2^{-n} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 2^{-n} \end{pmatrix}$$

A Biased Coin



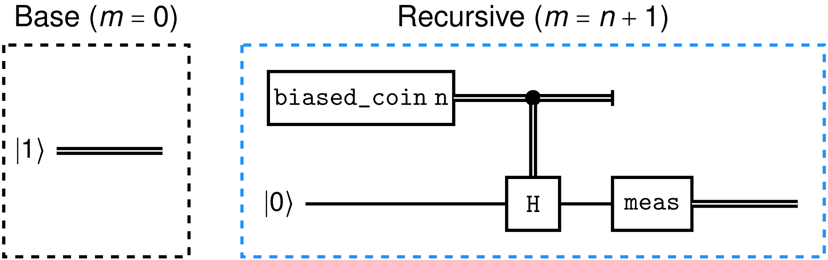

```
Fixpoint biased_coin (m : ℕ) : Box One Bit.
  box () ⇒ match m with
           | 0    ⇒ gate x ← new1 @(); output x
           | n+1  ⇒ let_ c      ← unbox (biased_coin n) ();
                    gate q      ← init0 @();
                    gate (c,q) ← bit_ctrl H @(c,q);
                    gate ()     ← discard @c;
                    gate b      ← meas @q;
                    output b
           end.
Defined.
```

A Biased Coin



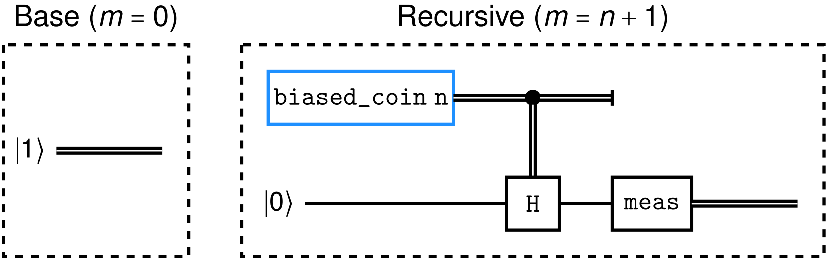

```
Fixpoint biased_coin (m : ℕ) : Box One Bit.
  box () ⇒ match m with
           | 0    ⇒ gate x ← new1 @(); output x
           | n+1  ⇒ let_ c      ← unbox (biased_coin n) ();
                    gate q      ← init0 @();
                    gate (c,q)  ← bit_ctrl H @(c,q);
                    gate ()     ← discard @c;
                    gate b      ← meas @q;
                    output b
           end.
Defined.
```

A Biased Coin



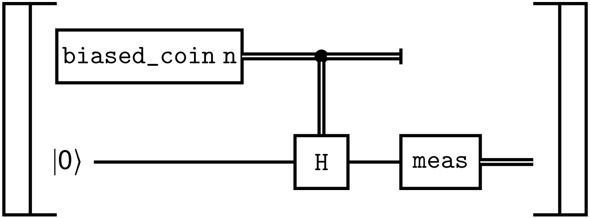

```
Fixpoint biased_coin (m : ℕ) : Box One Bit.
  box () ⇒ match m with
           | 0    ⇒ gate x ← new1 @(); output x
           | n+1  ⇒ let_ c      ← unbox (biased_coin n) ();
                    gate q      ← init0 @();
                    gate (c,q)  ← bit_ctrl H @(c,q);
                    gate ()     ← discard @c;
                    gate b      ← meas @q;
                    output b
           end.
Defined.
```

A Biased Coin

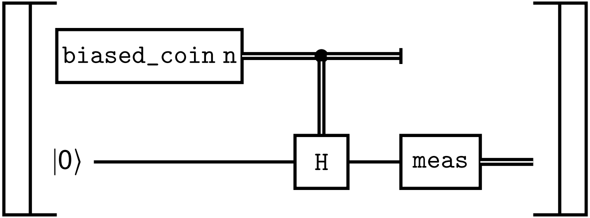



```
Fixpoint biased_coin (m : ℕ) : Box One Bit.
  box () ⇒ match m with
           | 0    ⇒ gate x ← new1 @(); output x
           | n+1  ⇒ let_ c     ← unbox (biased_coin n) ();
                    gate q     ← init0 @();
                    gate (c,q) ← bit_ctrl H @(c,q);
                    gate ()    ← discard @c;
                    gate b     ← meas @q;
                    output b
           end.
Defined.
```

A Biased Coin: Induction

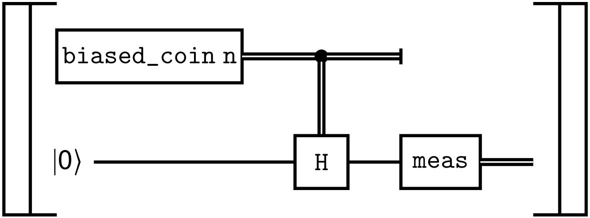


A Biased Coin: Induction



$$\begin{pmatrix} 1 - \frac{1}{2^{n+1}} & 0 \\ 0 & \frac{1}{2^{n+1}} \end{pmatrix}$$

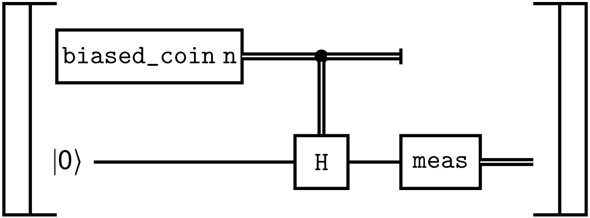
A Biased Coin: Induction



$$\begin{pmatrix} 1 - \frac{1}{2^{n+1}} & 0 \\ 0 & \frac{1}{2^{n+1}} \end{pmatrix}$$

$$[\![\texttt{biased\_coin 0}]\!] = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

A Biased Coin: Induction

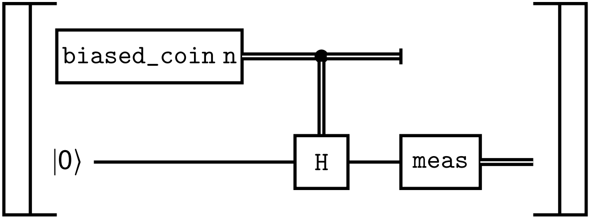


$$\begin{pmatrix} 1 - \frac{1}{2^{n+1}} & 0 \\ 0 & \frac{1}{2^{n+1}} \end{pmatrix}$$

$$[\![\texttt{biased\_coin 0}]\!] = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$[\![\texttt{biased\_coin (n+1)}]\!] = \mathit{meas}_2(\texttt{ctrl-H}([\![\texttt{biased\_coin n}]\!] \otimes |0\rangle\langle 0|))$$

A Biased Coin: Induction

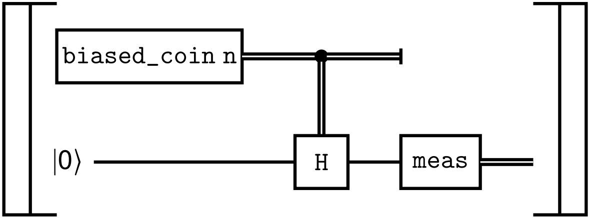


$$\begin{pmatrix} 1 - \frac{1}{2^{n+1}} & 0 \\ 0 & \frac{1}{2^{n+1}} \end{pmatrix}$$

$$[\![\texttt{biased\_coin 0}]\!] = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{aligned}[\![\texttt{biased\_coin (n+1)}]\!] &= \mathit{meas}_2(\texttt{ctrl-H}([\![\texttt{biased\_coin n}]\!] \otimes |0\rangle\langle 0|)) \\ &= \mathit{meas}_2(\texttt{ctrl-H}(\begin{pmatrix} 1-\frac{1}{2^n} & 0 \\ 0 & \frac{1}{2^n} \end{pmatrix} \otimes |0\rangle\langle 0|))\end{aligned}$$

A Biased Coin: Induction

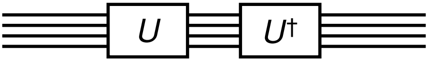


$$\begin{pmatrix} 1-\frac{1}{2^{n+1}} & 0 \\ 0 & \frac{1}{2^{n+1}} \end{pmatrix}$$

$$[\![\texttt{biased\_coin 0}]\!] = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

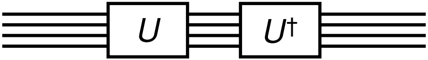
$$\begin{aligned} [\![\texttt{biased\_coin (n+1)}]\!] &= \mathit{meas}_2(\texttt{ctrl-H}([\![\texttt{biased\_coin n}]\!] \otimes |0\rangle\langle 0|)) \\ &= \mathit{meas}_2\left(\texttt{ctrl-H}\left(\begin{pmatrix} 1-\frac{1}{2^n} & 0 \\ 0 & \frac{1}{2^n} \end{pmatrix} \otimes |0\rangle\langle 0|\right)\right) \\ &= \begin{pmatrix} 1-\frac{1}{2^{n+1}} & 0 \\ 0 & \frac{1}{2^{n+1}} \end{pmatrix} \end{aligned}$$

Denoting Unitaries



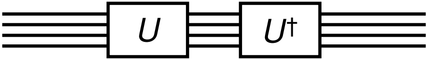

```
Definition unitary_transpose (n : ℕ) (U : Unitary n) :
  Box (n ⊗ Qubit) (n ⊗ Qubit).
  box ρ ⇒
    gate ρ ← U @ρ;
    gate ρ ← transpose U @ρ;
    output ρ.
Defined.
```

Denoting Unitaries



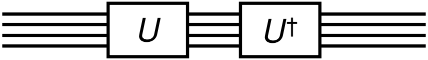

```
Lemma unitary_trans_id :
  ∀ (n : ℕ) (U : Unitary n) (ρ : Density_Matrix n),
  ⟦unitary_transpose U⟧ ρ = ρ.
```

Denoting Unitaries



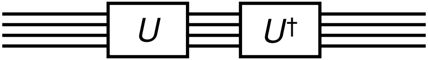
$$[\![\texttt{unitary\_transpose}]\!]\rho$$
$$= U^\dagger(U\rho U^\dagger)(U^\dagger)^\dagger$$

Denoting Unitaries



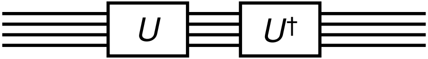
$$
\begin{aligned}
&[\![\texttt{unitary\_transpose}]\!]\rho \\
&= U^\dagger(U\rho U^\dagger)U
\end{aligned}
$$

Denoting Unitaries



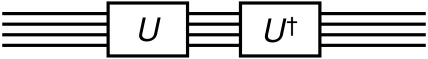
$$\llbracket \texttt{unitary\_transpose} \rrbracket \rho$$
$$= (U^\dagger U)\rho(U^\dagger U)$$

Denoting Unitaries



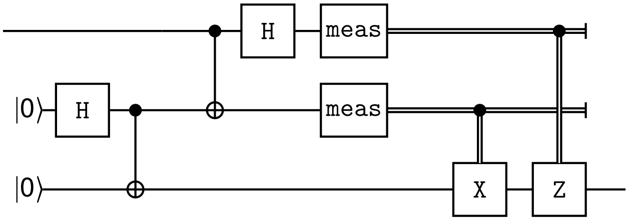
$$[\![\texttt{unitary\_transpose}]\!]\rho$$
$$= \mathbb{I}\rho\mathbb{I}$$

Denoting Unitaries



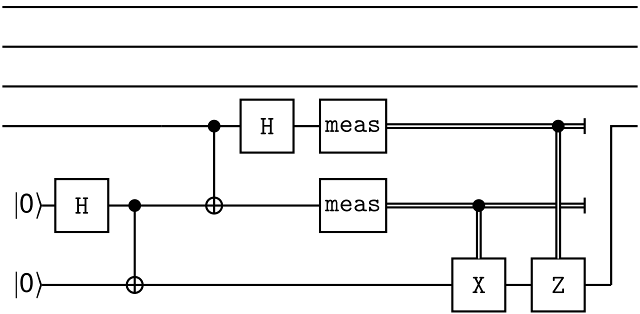
$$[\![\texttt{unitary\_transpose}]\!]\rho$$
$$= \rho$$

Quantum Teleportation




```
Lemma teleport_id : ∀ (ρ : Density_Matrix 1),
  ⟦teleport⟧ ρ = ρ.
```

Entangled Quantum Teleportation




```
Lemma teleport_id' : ∀ (n : ℕ) (ρ : Density_Matrix (n+1)),
  ⟦in_parallel (id n) teleport⟧ ρ = ρ.
```

Why $\mathcal{Q}$WIRE?

- Linearly typed circuits guarantee quantum realizability
- Dependent types allow precise specification of circuit families
- Embedded in Coq for classical control
- Denotational semantics for verified programming
- Categorical semantics (Renella and Staton, 2017)

Future Work

- Verified complex programs (QFT, Shor, Grover)
- Reversible circuits and verified compilation
  - as in ReVer (Amy *et al.*, 2017)
- Quantum oracles proven to correspond to classical functions

$\mathcal{FIN}$