Runtime Analysis of Quantum Programs

A Formal Approach

Federico Olmedo

Universidad de Chile
Chile

Alejandro Díaz-Caro

Universidad Nacional de Quilmes & ICC
Argentina

The 1st International Workshop on
Programming Languages for Quantum Computing

New Orleans, USA – January 2020

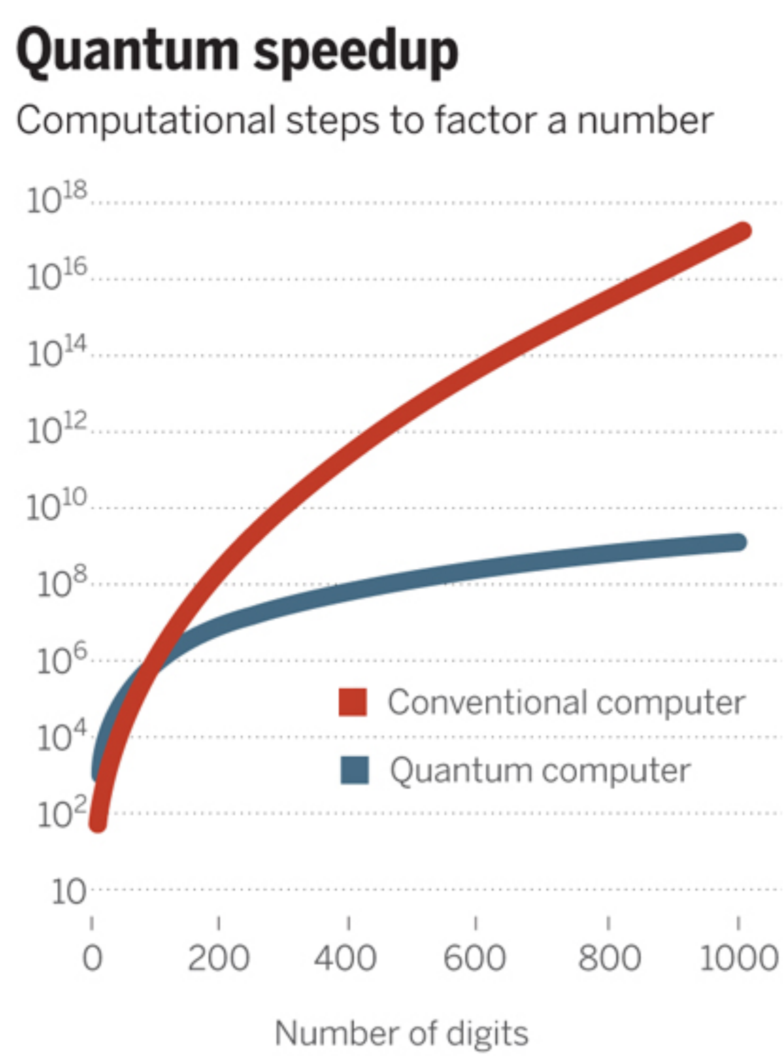
The role of quantum program resource analysis

Resource analysis is a particularly (more) relevant problem for quantum programs as it allows:

The role of quantum program resource analysis

Resource analysis is a particularly (more) relevant problem for quantum programs as it allows:

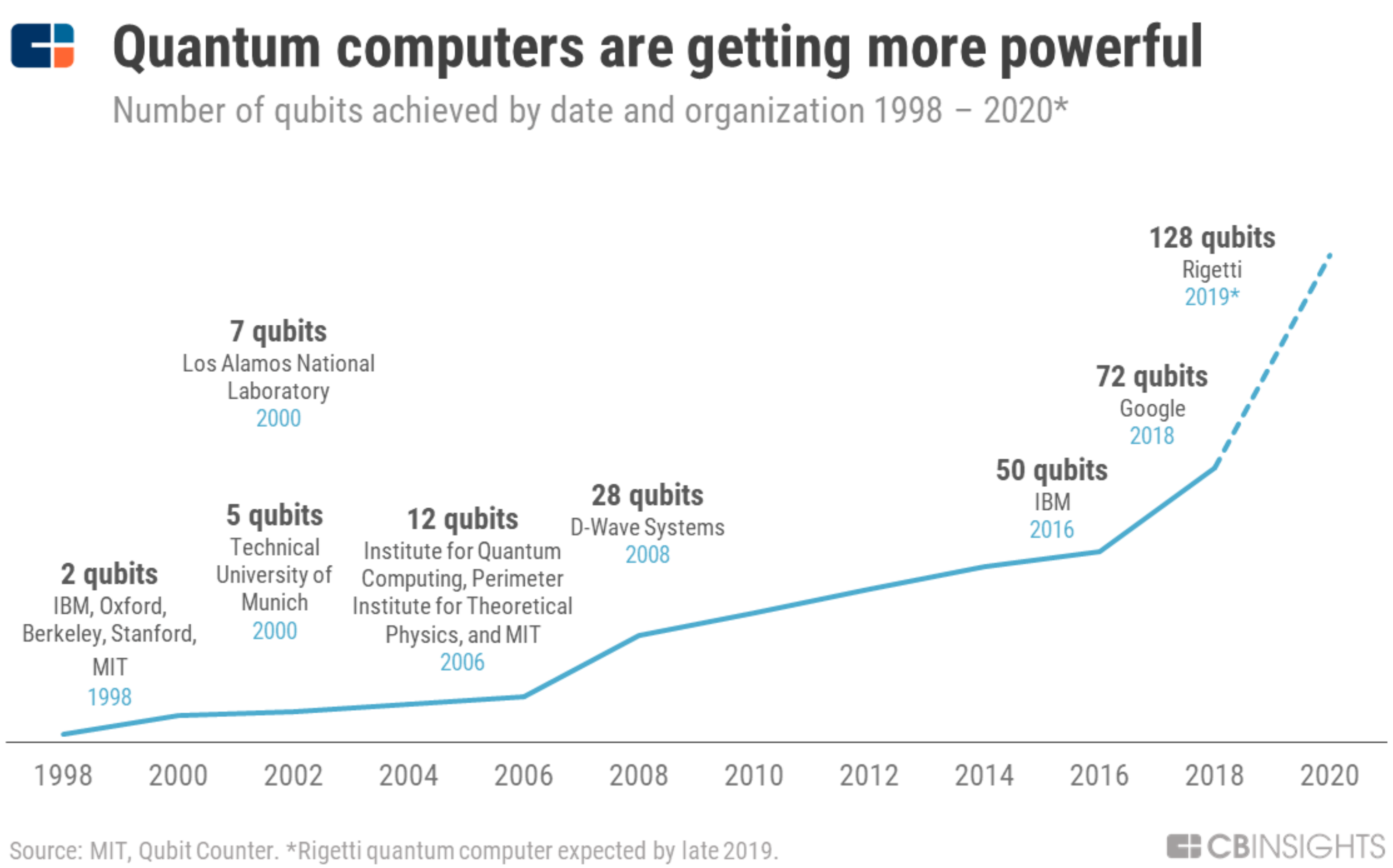
1. Validating the “effectivity” of the quantum computing model



The role of quantum program resource analysis

Resource analysis is a particularly (more) relevant problem for quantum programs as it allows:

2. Determining which quantum algorithms will be (shortly?) implementable in real quantum hardware



Current approach to quantum program resource analysis

[Quipper, Scaffold, LIQ*U*i|>]

1. Translate the (high-level) program into a (low-level) quantum circuit

```
#define n 1000
module foo(qbit q[n])
{
  for(int i=0;i<n;i++)
    H(q[i]);
  CNOT(q[n-1],q[0]);
}
module main()
{
  qbit b[n];
  foo(b);
}
```

```
qbit b[1000];
H ( b[0] );
H ( b[1] );
.
.
H ( b[999] );
CNOT ( b[999] , b[0] );
```

Current approach to quantum program resource analysis

[Quipper, Scaffold, LIQ*U*i|>]

1. Translate the (high-level) program into a (low-level) quantum circuit

```
#define n 1000
module foo(qbit q[n])
{
  for(int i=0;i<n;i++)
    H(q[i]);
  CNOT(q[n-1],q[0]);
}
module main()
{
  qbit b[n];
  foo(b);
}
```

```
qbit b[1000];
H ( b[0] );
H ( b[1] );
.
.
H ( b[999] );
CNOT ( b[999] , b[0] );
```

2. Read off the **number of qubits and gates** in the circuit

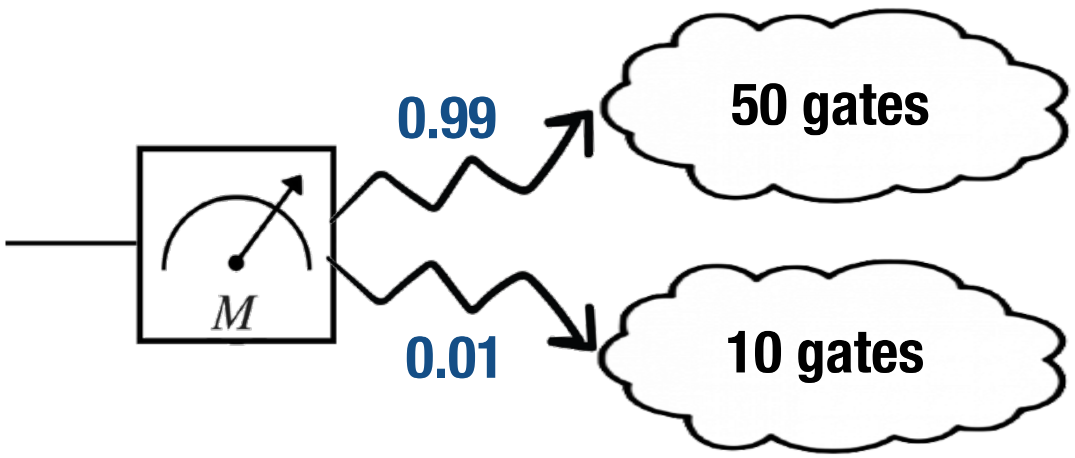
Resources				
Qubit	X	Z	H	T
2	400	27800	54300	55100

Current approach to quantum program resource analysis

Severe limitations

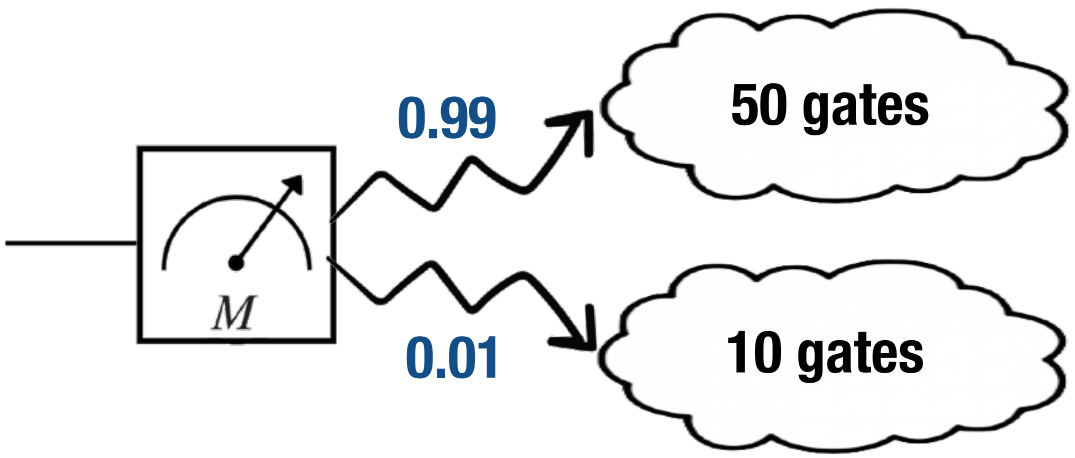
Current approach to quantum program resource analysis

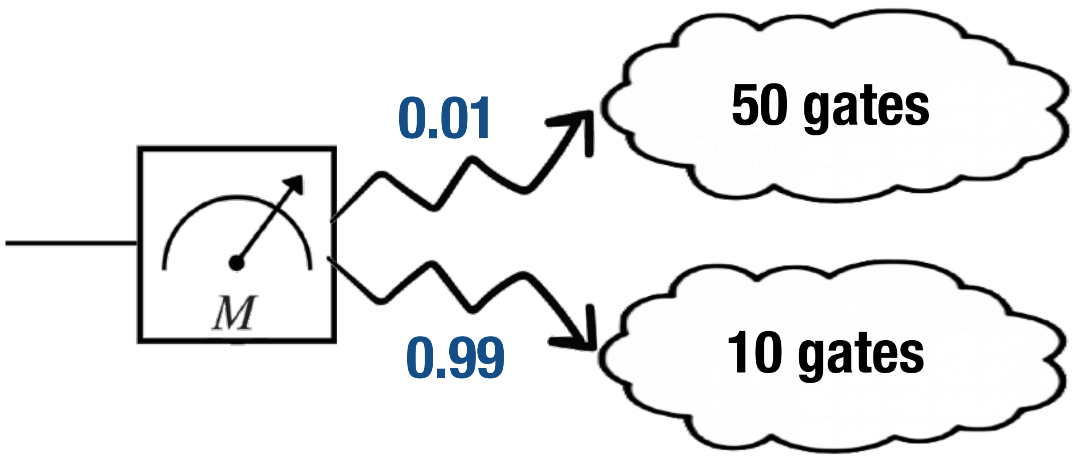
Severe limitations



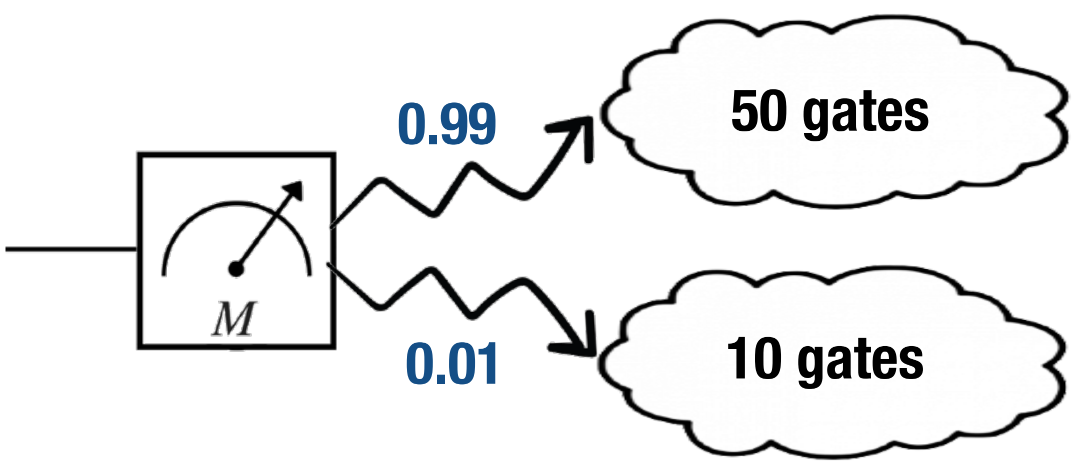
Current approach to quantum program resource analysis

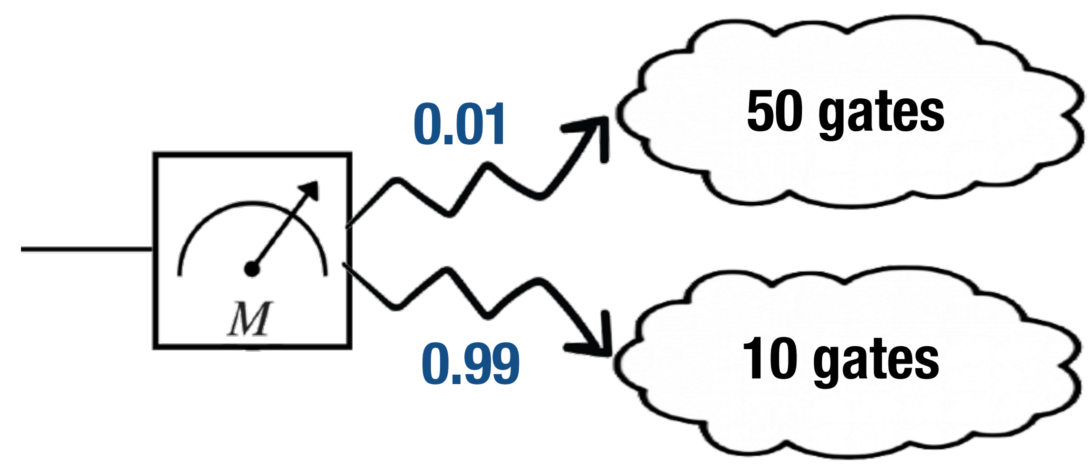
Severe limitations





Severe limitations



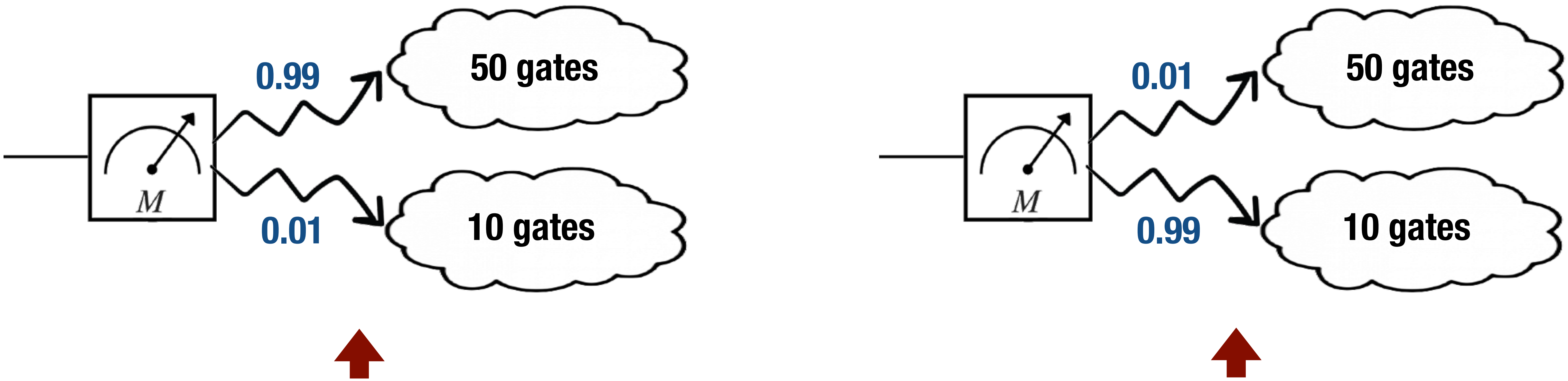


Same number of gates but (very!) different resource profiles

Current approach to quantum program resource analysis

Severe limitations

👎 Number of gates is a very poor resource measurement
(likelihood of execution paths is disregarded)



Same number of gates but (very!) different resource profiles

Current approach to quantum program resource analysis

Severe limitations

👎 Number of gates is a very poor resource measurement
(likelihood of execution paths is disregarded)

Current approach to quantum program resource analysis

Severe limitations

- 👎 Number of gates is a very poor resource measurement
  (likelihood of execution paths is disregarded)

- 👎 Restricted to programs with statically bounded loops

Current approach to quantum program resource analysis

Severe limitations

- 👎 Number of gates is a very poor resource measurement
  (likelihood of execution paths is disregarded)

- 👎 Restricted to programs with statically bounded loops

```
#define n 1000
module foo(qbit q[n])
{
  for(int i=0;i<n;i++)
    H(q[i]);
  CNOT(q[n-1],q[0]);
}
```

Current approach to quantum program resource analysis

Severe limitations

- Number of gates is a very poor resource measurement
  (likelihood of execution paths is disregarded)

- Restricted to programs with statically bounded loops

```
#define n 1000
module foo(qbit q[n])
{
  for(int i=0;i<n;i++)
                        prob. < 1
   ...
}
```

Current approach to quantum program resource analysis

Severe limitations

- Number of gates is a very poor resource measurement
  (likelihood of execution paths is disregarded)

- Restricted to programs with statically bounded loops

Sample algorithms out of scope

- BB84 quantum key distribution algorithm
- Simon's algorithm

```
#define n 1000
module foo(qbit q[n])
{
  for(int i=0;i<n;i++)

   ...
}
```

prob. < 1

Current approach to quantum program resource analysis

Severe limitations

👎 Number of gates is a very poor resource measurement
(likelihood of execution paths is disregarded)

👎 Restricted to programs with statically bounded loops

Sample algorithms out of scope

- BB84 quantum key distribution algorithm
- Simon's algorithm

Current approach to quantum program resource analysis

Severe limitations

- 👎 Number of gates is a very poor resource measurement
  (likelihood of execution paths is disregarded)

- 👎 Restricted to programs with statically bounded loops

  Sample algorithms out of scope

  - BB84 quantum key distribution algorithm
  - Simon's algorithm

- 👎 Rigid and low-level cost model

Our contribution

Calculus à la weakest precondition for reasoning about the runtime of quantum programs

- **Flexible:** accommodates multiple runtime models
- **Sensible:** accounts for execution probabilities
- **Expressive:** applies to programs with unbounded loops

Based on existing techniques for probabilistic programs
[Kaminski, Katoen, Matheja & Olmedo - ESOP'16, LICS'16, JACM 65:5]

The programming model (qGCL)

The programming model (qGCL)

Core imperative language over quantum variables with classical control flow

The programming model (qGCL)

Core imperative language over quantum variables with classical control flow

$c \ ::=$	$q := \|b\rangle$	varaible initialization

The programming model (qGCL)

Core imperative language over quantum variables with classical control flow

c	$::=$	$q := \|b\rangle$	varaible initialization
		$\overline{q} := U\,\overline{q}$	unitary transformation

The programming model (qGCL)

Core imperative language over quantum variables with classical control flow

$$
\begin{array}{lll}
c \ ::= & q := |b\rangle & \text{varaible initialization} \\
 & \overline{q} := U\,\overline{q} & \text{unitary transformation} \\
 & \square\ \mathcal{M}[\overline{q}] \ = \ \overline{m \rightarrow c_m} & \text{quantum case}
\end{array}
$$

The programming model (qGCL)

Core imperative language over quantum variables with classical control flow

c	$::=$	$q := \|b\rangle$	varaible initialization
		$\overline{q} := U\,\overline{q}$	unitary transformation
		$\Box\ \mathcal{M}[\overline{q}] = \overline{m \rightarrow c_m}$	quantum case
		while $(\mathcal{M}[\overline{q}] = 1)$ do c	quantum loop

The programming model (qGCL)

Core imperative language over quantum variables with classical control flow

c	$::=$	$q := \|b\rangle$	varaible initialization
		$\overline{q} := U\,\overline{q}$	unitary transformation
		$\square\ \mathcal{M}[\overline{q}]\ =\ \overline{m \rightarrow c_m}$	quantum case
		while $(\mathcal{M}[\overline{q}] = 1)$ do c	quantum loop
		skip	no-op
		$c_1 ; c_2$	sequential composition

Our approach to the runtime of quantum programs

Our approach to the runtime of quantum programs

set of program states

GOAL:

$$[\![c]\!]^{⧗} : \mathcal{S} \to \mathbb{R}_{\geq 0}^{\infty}$$

$$[\![c]\!]^{⧗}\, \rho \quad = \quad \text{runtime of } c \text{ from initial state } \rho$$

⧗

Our approach to the runtime of quantum programs

set of program states

GOAL:

$$[\![c]\!]^{⧗} : \ \mathcal{S} \to \mathbb{R}_{\geq 0}^{\infty}$$

$$[\![c]\!]^{⧗}\ \rho \ = \ \text{runtime of } c \text{ from initial state } \rho$$

APPROACH: Continuation passing style through **runtime transformer**

$$\mathsf{ert}\,[c] : \ (\mathcal{S} \to \mathbb{R}_{\geq 0}^{\infty}) \to (\mathcal{S} \to \mathbb{R}_{\geq 0}^{\infty})$$

⧗

Our approach to the runtime of quantum programs

GOAL:

set of program states

$$[\![c]\!]^{⧗} : \; \mathcal{S} \to \mathbb{R}^{\infty}_{\geq 0}$$

$$[\![c]\!]^{⧗} \, \rho \;=\; \text{runtime of } c \text{ from initial state } \rho$$

APPROACH: Continuation passing style through runtime transformer

$$\mathrm{ert}\,[c] : \; (\mathcal{S} \to \mathbb{R}^{\infty}_{\geq 0}) \to (\mathcal{S} \to \mathbb{R}^{\infty}_{\geq 0})$$

runtime of the program following c

⧗

Our approach to the runtime of quantum programs

GOAL:

set of program states

$$[\![c]\!]^{⧗} : \; \mathcal{S} \rightarrow \mathbb{R}^{\infty}_{\geq 0}$$

$$[\![c]\!]^{⧗} \; \rho \; = \; \text{runtime of } c \text{ from initial state } \rho$$

APPROACH: Continuation passing style through **runtime transformer**

$$\mathrm{ert}\,[c] : \; (\mathcal{S} \rightarrow \mathbb{R}^{\infty}_{\geq 0}) \rightarrow (\mathcal{S} \rightarrow \mathbb{R}^{\infty}_{\geq 0})$$

runtime of the program following c

runtime of c, plus the program following c

⧗

Our approach to the runtime of quantum programs

GOAL:

set of program states

$$[\![c]\!]^{\text{⧗}} : \mathcal{S} \to \mathbb{R}_{\geq 0}^{\infty}$$

$$[\![c]\!]^{\text{⧗}}\,\rho \quad = \quad \text{runtime of } c \text{ from initial state } \rho$$

APPROACH:

Continuation passing style through **runtime transformer**

$$\mathrm{ert}\,[c] : \left(\mathcal{S} \to \mathbb{R}_{\geq 0}^{\infty}\right) \to \left(\mathcal{S} \to \mathbb{R}_{\geq 0}^{\infty}\right)$$

runtime of the program following c

runtime of c, plus the program following c

In particular,

$$\mathrm{ert}\,[c]\,(\lambda \rho'.\, 0) \quad = \quad [\![c]\!]^{\text{⧗}}$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\mathrm{ert}[\overline{q} := U\,\overline{q}](t) =$$

$$\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \rightarrow c_m}](t) =$$

$$\mathrm{ert}[c_1; c_2](t) =$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\overline{q} := U\,\overline{q} \quad ; \quad \cdots\cdots$$

$$\mathrm{ert}[\overline{q} := U\,\overline{q}](t) \quad =$$

$$\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \rightarrow c_m}](t) \quad =$$

$$\mathrm{ert}[c_1; c_2](t) \quad =$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\overline{q} := U\,\overline{q} \quad ; \quad \cdots\cdots$$

t

$$\mathrm{ert}[\overline{q} := U\,\overline{q}](t) =$$

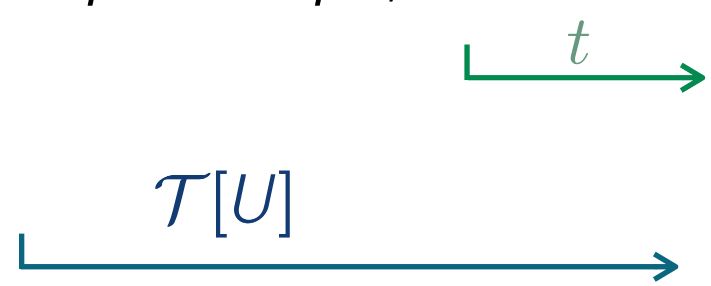
$$\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \to c_m}](t) =$$

$$\mathrm{ert}[c_1; c_2](t) =$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\overline{q} := U\,\overline{q} \quad ; \quad \cdots\cdots$$



$\mathsf{ert}[\overline{q} := U\,\overline{q}](t) \quad =$

$\mathsf{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \to c_m}](t) \quad =$

$\mathsf{ert}[c_1; c_2](t) \quad =$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\overline{q} := U\,\overline{q} \quad ; \quad \cdots\cdots$$

$$t$$

$$\mathcal{T}[U] \;+\; t$$

$$\begin{aligned}
\mathrm{ert}[\overline{q} := U\,\overline{q}](t) &= \\
\mathrm{ert}[\square\; \mathcal{M}[\overline{q}] \;=\; \overline{m \rightarrow c_m}](t) &= \\
\mathrm{ert}[c_1; c_2](t) &=
\end{aligned}$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\overline{q} := U\,\overline{q} \quad ; \quad \cdots\cdots$$

t

$$\mathcal{T}[U] \quad + \quad t \circ [\![U]\!]$$

$$\begin{aligned}
\mathsf{ert}[\overline{q} := U\,\overline{q}](t) &= \\
\mathsf{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \rightarrow c_m}](t) &= \\
\mathsf{ert}[c_1; c_2](t) &=
\end{aligned}$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\overline{q} := U\,\overline{q} \;\; ; \quad \cdots\cdots$$

$$t$$

$$\mathcal{T}[U] \;+\; t \circ [\![U]\!]$$

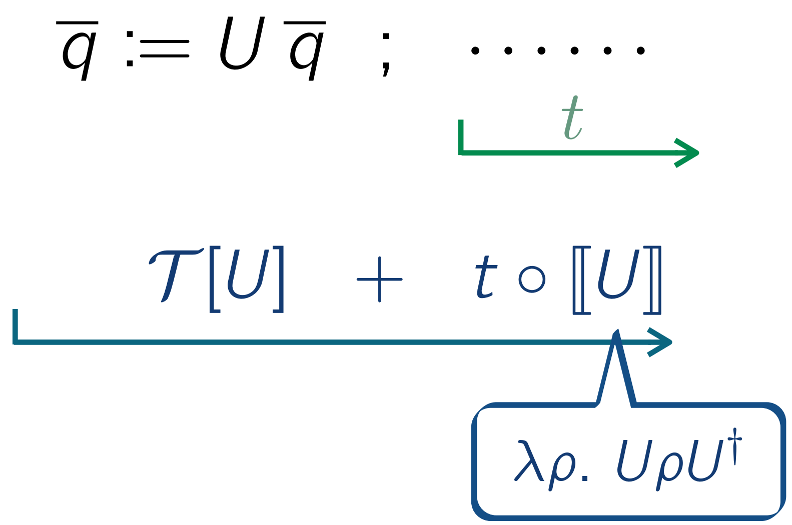
$$\lambda\rho.\; U\rho U^{\dagger}$$

$\text{ert}[\overline{q} := U\,\overline{q}](t) \quad =$

$\text{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \rightarrow c_m}](t) \quad =$

$\text{ert}[c_1;\, c_2](t) \quad =$

Transformer ert[c] admits an elegant **definition by induction** on the structure of c



$$\text{ert}[\overline{q} := U\,\overline{q}](t) = \mathcal{T}[U] + t \circ [\![U]\!]$$

$$\text{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \to c_m}](t) =$$

$$\text{ert}[c_1; c_2](t) =$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\mathrm{ert}[\overline{q} := U\,\overline{q}](t) \quad = \quad \mathcal{T}[U] + t \circ [\![U]\!]$$

$$\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \rightarrow c_m}](t) \quad =$$

$$\mathrm{ert}[c_1; c_2](t) \quad =$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\Box\ \mathcal{M}[\overline{q}] \ = \ m_1 \to c_1 \mid m_2 \to c_2 \quad ; \quad \ldots\ldots$$

$$\begin{aligned}
\mathrm{ert}[\overline{q} := U\ \overline{q}](t) \quad &= \quad \mathcal{T}[U] \ + \ t \circ [\![U]\!] \\
\mathrm{ert}[\Box\ \mathcal{M}[\overline{q}] \ = \ \overline{m \to c_m}](t) \quad &= \\
\mathrm{ert}[c_1; c_2](t) \quad &=
\end{aligned}$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\square\ \mathcal{M}[\overline{q}] \;=\; m_1 \to c_1 \mid m_2 \to c_2 \quad ; \quad \ldots\ldots$$

t

$$\mathrm{ert}[\overline{q} := U\ \overline{q}](t) \quad = \quad \mathcal{T}[U] \;+\; t \circ [\![U]\!]$$

$$\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] \;=\; \overline{m \to c_m}](t) \quad =$$

$$\mathrm{ert}[c_1; c_2](t) \quad =$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\square\ \mathcal{M}[\overline{q}] = m_1 \to c_1 \mid m_2 \to c_2 \ ; \quad \ldots\ldots$$

t

$\mathcal{T}[\mathcal{M}]$

$$\begin{aligned}
\text{ert}[\overline{q} := U\,\overline{q}](t) &= \mathcal{T}[U] + t \circ [\![U]\!] \\
\text{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \to c_m}](t) &= \\
\text{ert}[c_1; c_2](t) &=
\end{aligned}$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\square\ \mathcal{M}[\overline{q}] = m_1 \to c_1 \mid m_2 \to c_2 \quad ; \quad \ldots\ldots$$

t

$$\mathcal{T}[\mathcal{M}] + \qquad \text{ert}[c_1](t) \qquad + \qquad \text{ert}[c_2](t)$$

$$\begin{aligned}
\text{ert}[\overline{q} := U\,\overline{q}](t) &= \mathcal{T}[U] + t \circ [\![U]\!] \\
\text{ert}[\square\ \mathcal{M}[\overline{q}] = \overline{m \to c_m}](t) &= \\
\text{ert}[c_1; c_2](t) &=
\end{aligned}$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\square\ \mathcal{M}[\overline{q}] \;=\; m_1 \to c_1 \mid m_2 \to c_2 \;\; ; \quad \ldots\ldots$$

$$t$$

$$\mathcal{T}[\mathcal{M}] \;+\; \mathrm{ert}[c_1](t) \circ [\![\mathcal{M}{=}m_1]\!] \;+\; \mathrm{ert}[c_2](t) \circ [\![\mathcal{M}{=}m_2]\!]$$

$$\mathrm{ert}[\overline{q} := U\,\overline{q}](t) \;=\; \mathcal{T}[U] \;+\; t \circ [\![U]\!]$$

$$\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] \;=\; \overline{m \to c_m}](t) \;=$$

$$\mathrm{ert}[c_1; c_2](t) \;=$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\square\ \mathcal{M}[\overline{q}] \ = \ m_1 \rightarrow c_1 \mid m_2 \rightarrow c_2 \ \ ; \quad \ldots\ldots$$

t

$$\mathcal{T}[\mathcal{M}] \ + \ \Pr[\mathcal{M}{=}m_1] \cdot \mathrm{ert}[c_1](t) \circ [\![\mathcal{M}{=}m_1]\!] \ + \ \Pr[\mathcal{M}{=}m_2] \cdot \mathrm{ert}[c_2](t) \circ [\![\mathcal{M}{=}m_2]\!]$$

$$\begin{aligned}
\mathrm{ert}[\overline{q} := U\,\overline{q}](t) \quad &= \quad \mathcal{T}[U] \ + \ t \circ [\![U]\!] \\
\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] \ = \ \overline{m \rightarrow c_m}](t) \quad &= \\
\mathrm{ert}[c_1; c_2](t) \quad &=
\end{aligned}$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\square\ \mathcal{M}[\overline{q}] \ = \ m_1 \rightarrow c_1 \mid m_2 \rightarrow c_2 \ \ ; \quad \ldots\ldots$$

t

$$\mathcal{T}[\mathcal{M}] \ + \ \Pr[\mathcal{M}{=}m_1] \cdot \mathrm{ert}[c_1](t) \circ [\![\mathcal{M}{=}m_1]\!] \ + \ \Pr[\mathcal{M}{=}m_2] \cdot \mathrm{ert}[c_2](t) \circ [\![\mathcal{M}{=}m_2]\!]$$

$$\mathrm{ert}[\overline{q} := U\, \overline{q}](t) \quad = \quad \mathcal{T}[U] \ + \ t \circ [\![U]\!]$$

$$\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] \ = \ \overline{m \rightarrow c_m}](t) \quad = \quad \mathcal{T}[\mathcal{M}] \ + \ \textstyle\sum_m \Pr[\mathcal{M}{=}m] \cdot \mathrm{ert}[c_m](t) \circ [\![\mathcal{M}[\overline{q}]]\!]$$

$$\mathrm{ert}[c_1; c_2](t) \quad =$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\square\ \mathcal{M}[\overline{q}] \ = \ m_1 \to c_1 \mid m_2 \to c_2 \ \ ; \quad \ldots\ldots$$

$$t$$

$$\mathcal{T}[\mathcal{M}] \ + \ \Pr[\mathcal{M}{=}m_1]\cdot \mathrm{ert}[c_1](t) \circ [\![\mathcal{M}{=}m_1]\!] \ + \ \Pr[\mathcal{M}{=}m_2]\cdot \mathrm{ert}[c_2](t) \circ [\![\mathcal{M}{=}m_2]\!]$$

$$
\begin{aligned}
\mathrm{ert}[\overline{q} := U\ \overline{q}](t) \quad &= \quad \mathcal{T}[U] \ + \ t \circ [\![U]\!] \\
\mathrm{ert}[\square\ \mathcal{M}[\overline{q}] \ = \ \overline{m \to c_m}](t) \quad &= \quad \mathcal{T}[\mathcal{M}] \ + \ \textstyle\sum_m \Pr[\mathcal{M}{=}m]\cdot \mathrm{ert}[c_m](t) \circ [\![\mathcal{M}[\overline{q}]]\!] \\
\mathrm{ert}[c_1; c_2](t) \quad &= \quad \mathrm{ert}[c_1](\mathrm{ert}[c_2](t))
\end{aligned}
$$

Definition of runtime transformer ert

Transformer ert[c] admits an elegant **definition by induction** on the structure of c

$$\square\ \mathcal{M}[\overline{q}] \ = \ m_1 \to c_1 \mid m_2 \to c_2 \ \ ; \quad \ldots\ldots$$

$$t$$

$$\mathcal{T}[\mathcal{M}] \ + \ \mathsf{Pr}[\mathcal{M}{=}m_1] \cdot \mathsf{ert}[c_1](t) \circ [\![\mathcal{M}{=}m_1]\!] \ + \ \mathsf{Pr}[\mathcal{M}{=}m_2] \cdot \mathsf{ert}[c_2](t) \circ [\![\mathcal{M}{=}m_2]\!]$$

$$\begin{aligned}
\mathsf{ert}[\overline{q} := U\ \overline{q}](t) \quad &= \quad \mathcal{T}[U] \ + \ t \circ [\![U]\!] \\
\mathsf{ert}[\square\ \mathcal{M}[\overline{q}] \ = \ \overline{m \to c_m}](t) \quad &= \quad \mathcal{T}[\mathcal{M}] \ + \ \textstyle\sum_m \mathsf{Pr}[\mathcal{M}{=}m] \cdot \mathsf{ert}[c_m](t) \circ [\![\mathcal{M}[\overline{q}]]\!] \\
\mathsf{ert}[c_1; c_2](t) \quad &= \quad \mathsf{ert}[c_1](\mathsf{ert}[c_2](t)) \\
&\ \ \vdots
\end{aligned}$$

Invariant-based reasoning for the runtime of loops

We can establish upper bounds for the runtime of loops using a notion of **loop invariant**:

Invariant-based reasoning for the runtime of loops

We can establish upper bounds for the runtime of loops using a notion of **loop invariant**:

$$\frac{F_t^{\langle \mathcal{M}, c\rangle}(I) \;\preceq\; I}{\text{ert}[\text{while } (\mathcal{M} = 1) \text{ do } c](t) \;\preceq\; I}$$

← *I* is a loop invariant

← *I* is an upper bound of the loop runtime

Case study: BB84 quantum key distribution algorithm

Case study: BB84 quantum key distribution algorithm

GOAL: securely create and distribute a shared (symmetric) key between two parties.

Case study: BB84 quantum key distribution algorithm

GOAL: securely create and distribute a shared (symmetric) key between two parties.

```
\\ initialize counter
k := |0⟩;
\\ while not reached m bits
while (M[k] = 1) do
   \\ flip Alice's and Bob's coins
   A := |++⟩; B := |+⟩;
   \\ measure Alice's coins
   □ · M_A[A] =
               \\ measure Bob's coin
     |eb⟩ → □ · M_B[B]
               \\ if Alice's and Bob's basis agree
               \\ store bit b and increment counter
               |e⟩ → k, Q := U_{P_b}[k, Q];
                          k := U_>[k];
               \\ if Alice's and Bob's basis disagree
               \\ discard bit b
               |¬e⟩ → skip
```

Case study: BB84 quantum key distribution algorithm

GOAL: securely create and distribute a shared (symmetric) key between two parties.

```
\\ initialize counter
k := |0⟩;
\\ while not reached m bits
while (M[k] = 1) do
   \\ flip Alice's and Bob's coins
   A := |++⟩; B := |+⟩;
   \\ measure Alice's coins
   □ · M_A[A] =
            \\ measure Bob's coin
     |eb⟩ → □ · M_B[B]
              \\ if Alice's and Bob's basis agree
              \\ store bit b and increment counter
              |e⟩ → k, Q := U_{P_b}[k, Q];
                    k := U_>[k];
              \\ if Alice's and Bob's basis disagree
              \\ discard bit b
              |¬e⟩ → skip
```

Average time required to generate a key of *m* bits:

$$\mathcal{T}[|0\rangle] + 2\mathcal{T}m + \mathcal{T}[\mathcal{M}] \quad \in \mathcal{O}(m)$$

Case study: BB84 quantum key distribution algorithm

GOAL: securely create and distribute a shared (symmetric) key between two parties.

\\\\ *initialize counter*
$k := |0\rangle;$
\\\\ *while not reached m bits*
while $(\mathcal{M}[k] = 1)$ do
\\\\ *flip Alice's and Bob's coins*
$A := |{+}{+}\rangle;\ B := |{+}\rangle;$
\\\\ *measure Alice's coins*
$\square \cdot \mathcal{M}_A[A]\ =$
\\\\ *measure Bob's coin*
$|eb\rangle \rightarrow \square \cdot \mathcal{M}_B[B]$
\\\\ *if Alice's and Bob's basis agree*
\\\\ *store bit b and increment counter*
$|e\rangle \rightarrow k, Q := U_{P_b}[k, Q];$
$k := U_{>}[k];$
\\\\ *if Alice's and Bob's basis disagree*
\\\\ *discard bit b*
$|\neg e\rangle \rightarrow$ skip

Average time required to generate a key of *m* bits:

$$\mathcal{T}[|0\rangle] + 2\mathcal{T}m + \mathcal{T}[\mathcal{M}] \in \mathcal{O}(m)$$

counter *k* initialization

Case study: BB84 quantum key distribution algorithm

GOAL: securely create and distribute a shared (symmetric) key between two parties.

```
\\ initialize counter
k := |0⟩;
\\ while not reached m bits
while (M[k] = 1) do
   \\ flip Alice's and Bob's coins
   A := |++⟩; B := |+⟩;
   \\ measure Alice's coins
   □ · M_A[A] =
              \\ measure Bob's coin
     |eb⟩ → □ · M_B[B]
                \\ if Alice's and Bob's basis agree
                \\ store bit b and increment counter
                |e⟩ → k, Q := U_{P_b}[k, Q];
                      k := U_>[k];
                \\ if Alice's and Bob's basis disagree
                \\ discard bit b
                |¬e⟩ → skip
```

Average time required to generate a key of *m* bits:

$$\mathcal{T}[|0\rangle] + 2\mathcal{T}m + \mathcal{T}[\mathcal{M}] \quad \in \mathcal{O}(m)$$

counter k initialization

loop body

Case study: BB84 quantum key distribution algorithm

GOAL: securely create and distribute a shared (symmetric) key between two parties.

\\ initialize counter
$k := |0\rangle;$
\\ while not reached m bits
while $(\mathcal{M}[k] = 1)$ do
  \\ flip Alice's and Bob's coins
  $A := |{+}{+}\rangle;\ B := |{+}\rangle;$
  \\ measure Alice's coins
  $\square \cdot \mathcal{M}_A[A]\ =$
    \\ measure Bob's coin
    $|eb\rangle \rightarrow \square \cdot \mathcal{M}_B[B]$
      \\ if Alice's and Bob's basis agree
      \\ store bit b and increment counter
      $|e\rangle \rightarrow k, Q := U_{P_b}[k, Q];$
        $k := U_{>}[k];$
      \\ if Alice's and Bob's basis disagree
      \\ discard bit b
      $|\neg e\rangle \rightarrow$ skip

Average time required to generate a key of *m* bits:

$$\mathcal{T}[|0\rangle] + 2\mathcal{T}m + \mathcal{T}[\mathcal{M}] \quad \in \mathcal{O}(m)$$

counter k initialization

loop body

Final measurement upon loop exit

Case study: BB84 quantum key distribution algorithm

GOAL: securely create and distribute a shared (symmetric) key between two parties.

\\ initialize counter
$k := |0\rangle;$
\\ while not reached m bits
while $(\mathcal{M}[k] = 1)$ do
  \\ flip Alice's and Bob's coins
  $A := |{+}{+}\rangle;\ B := |{+}\rangle;$
  \\ measure Alice's coins
  $\square \cdot \mathcal{M}_A[A] \ =$
      \\ measure Bob's coin
    $|eb\rangle \rightarrow \square \cdot \mathcal{M}_B[B]$
      \\ if Alice's and Bob's basis agree
      \\ store bit b and increment counter
      $|e\rangle \rightarrow k, Q := U_{P_b}[k, Q];$
        $k := U_{\succ}[k];$
      \\ if Alice's and Bob's basis disagree
      \\ discard bit b
      $|\neg e\rangle \rightarrow$ skip

Average time required to generate a key of *m* bits:

$$\mathcal{T}[|0\rangle] + 2\mathcal{T}m + \mathcal{T}[\mathcal{M}] \quad \in \mathcal{O}(m)$$

counter *k* initialization

loop body

Final measurement upon loop exit

$$\mathcal{T} \ = \ \mathcal{T}[\mathcal{M}] + \mathcal{T}[|{+}{+}\rangle] + \mathcal{T}[|{+}\rangle] + \mathcal{T}[\mathcal{M}_A] + \mathcal{T}[\mathcal{M}_B] + \tfrac{1}{2}\left(\tfrac{1}{2}\mathcal{T}[U_{P_0}] + \tfrac{1}{2}\mathcal{T}[U_{P_1}] + \mathcal{T}[U_{\succ}]\right) + \tfrac{1}{2}$$

Conclusion

Conclusion

First step to a *formal and compelling* resource analysis of quantum programs

- Existing techniques for probabilistic programs extend smoothly to quantum programs

Conclusion

First step to a *formal and compelling* resource analysis of quantum programs

➡ Existing techniques for probabilistic programs extend smoothly to quantum programs

Future work

- Connection to an operational model
- Language extensions
- Automation

Conclusion

First step to a *formal and compelling*
resource analysis of quantum programs

➡ Existing techniques for probabilistic programs extend smoothly to quantum programs

Future work

- Connection to an operational model
- Language extensions
- Automation

Thanks!

BACKUP SLIDES

Language semantics

$$
\begin{aligned}
[\![\mathrm{skip}]\!](\rho) &= \rho \\
[\![q := |b\rangle]\!](\rho) &= \rho[q \mapsto |b\rangle] \\
[\![\overline{q} := U\ \overline{q}]\!](\rho) &= U^{\uparrow q} \rho {U^{\uparrow q}}^{\dagger} \\
[\![c_1 ; c_2]\!](\rho) &= [\![c_1]\!]([\![c_2]\!](\rho)) \\
[\![\square\ \mathcal{M}[\overline{q}]\ =\ \overline{m \to c_m}]\!](\rho) &= \textstyle\sum_m \mathrm{Pr}_\rho[\mathcal{M}{=}m] \cdot [\![c_m]\!](\rho|_{\mathcal{M}=m}) \\
[\![\mathrm{while}\ (\mathcal{M}[\overline{q}] = 1)\ \mathrm{do}\ c]\!](\rho) &= \mathit{lfp}\big(\Phi^{\langle \mathcal{M}, c\rangle}\big)
\end{aligned}
$$

$\mathrm{tr}({M_m}^{\dagger} M_m \rho)$

$\dfrac{M_m \rho {M_m}^{\dagger}}{\mathrm{tr}({M_m}^{\dagger} M_m \rho)}$

$\lambda X.\ \lambda \rho'.\ \mathrm{Pr}_{\rho'}[\mathcal{M}{=}0] \cdot \rho'|_{\mathcal{M}=0} + X\big(\mathrm{Pr}_{\rho'}[\mathcal{M}{=}1] \cdot [\![c]\!](\rho'|_{\mathcal{M}=1})\big)$

Full definition of ert transformer

$$
\begin{aligned}
\text{ert}[\text{skip}](t) &= \lambda\rho.\ 1 + t(\rho) \\
\text{ert}[q := |b\rangle](t) &= \lambda\rho.\ \mathcal{T}[|b\rangle] + t\big(\rho[q \mapsto |b\rangle]\big) \\
\text{ert}[\overline{q} := U\ \overline{q}](t) &= \lambda\rho.\ \mathcal{T}[U] + t\Big(U^{\uparrow q}\rho {U^{\uparrow q}}^{\dagger}\Big) \\
\text{ert}[c_1; c_2](t) &= \text{ert}[c_1](\text{ert}[c_2](t)) \\
\text{ert}[\square\ \mathcal{M}[\overline{q}]\ =\ \overline{m \to c_m}](t) &= \lambda\rho.\ \mathcal{T}[\mathcal{M}] + \textstyle\sum_m \text{Pr}_\rho[\mathcal{M}{=}m] \cdot \text{ert}[c_m](t)\big(\rho|_{\mathcal{M}=m}\big) \\
\text{ert}[\text{while } (\mathcal{M}[\overline{q}] = 1) \text{ do } c](t) &= \mathit{lfp}\big(F_t^{\langle \mathcal{M}[\overline{q}], c\rangle}\big)
\end{aligned}
$$

$$\lambda t'.\ \lambda\rho.\ \mathcal{T}[\mathcal{M}] + \text{Pr}_\rho[\mathcal{M}{=}1] \cdot \text{ert}[c](t')(\rho|_{\mathcal{M}=1}) + \text{Pr}_\rho[\mathcal{M}{=}0] \cdot t(\rho|_{\mathcal{M}=0})$$