Silq: A High-level Quantum Programming Language

Benjamin Bichsel

Maximilian Baader

Timon Gehr

Martin Vechev

SRILAB

ETH zürich

Quantum Programming Languages

Existing languages: Describe circuits

- Qiskit
- Quipper
- Q#
- ProjectQ
- Cirq
- QWire
- LIQ$Ui|\rangle$
- QASM
- Forest
- ...

Silq: Describes high-level operations

Reduce & simplify code

Static safety / Prevent errors

Safe automatic uncomputation

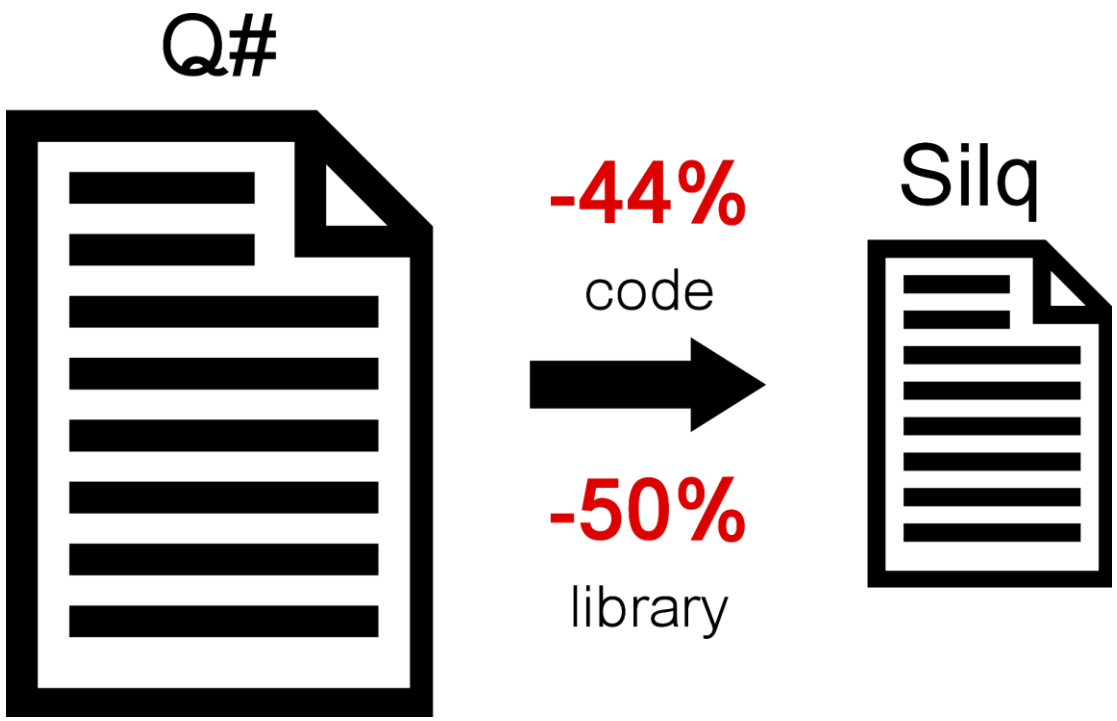
More intuitive semantics

Physicality

Downstream applications

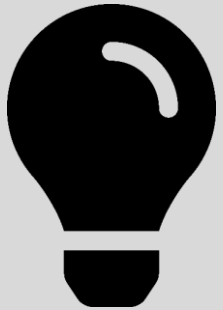
Source of icons: https://fontawesome.com

Quantum Programming Languages

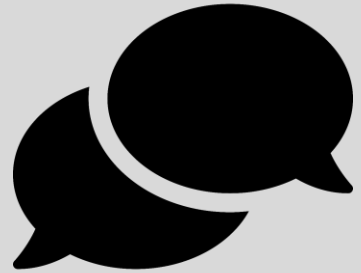


Comparison on Microsoft's Q# coding contests, see https://silq.ethz.ch/comparison

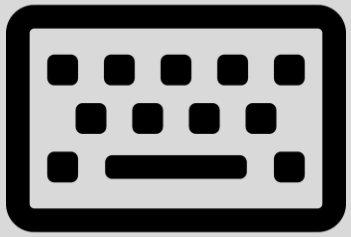
Goals of this Lecture



Learn Silq's language concepts



Discuss & compare languages



Try out Silq

Report issues

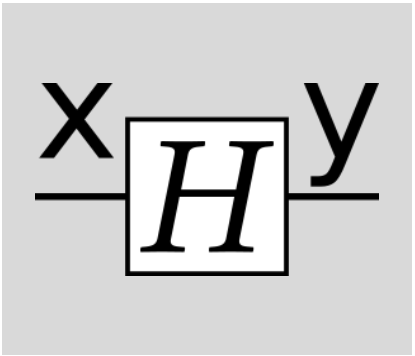
Source of icons: https://fontawesome.com

Background (minimal)

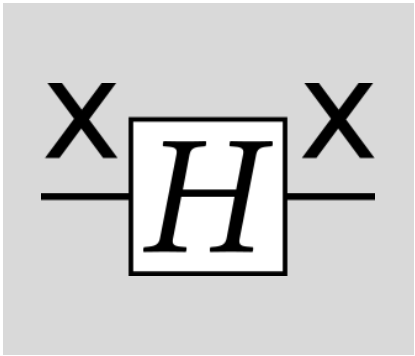
Concept	Explanation
State of quantum bit	$\varphi = \gamma_0 \lvert 0\rangle + \gamma_1 \lvert 1\rangle$ for $\gamma_0, \gamma_1 \in \mathbb{C}$
(Computational) basis states	$\lvert 0\rangle, \lvert 1\rangle$, but not $\frac{1}{\sqrt{2}} \lvert 0\rangle + \frac{1}{\sqrt{2}} \lvert 1\rangle$
No-cloning theorem	Impossible: $\varphi \longmapsto \varphi \otimes \varphi$

Basic Features of Silq

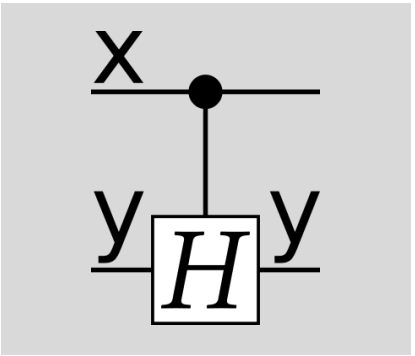
```
y:=H(x);
```



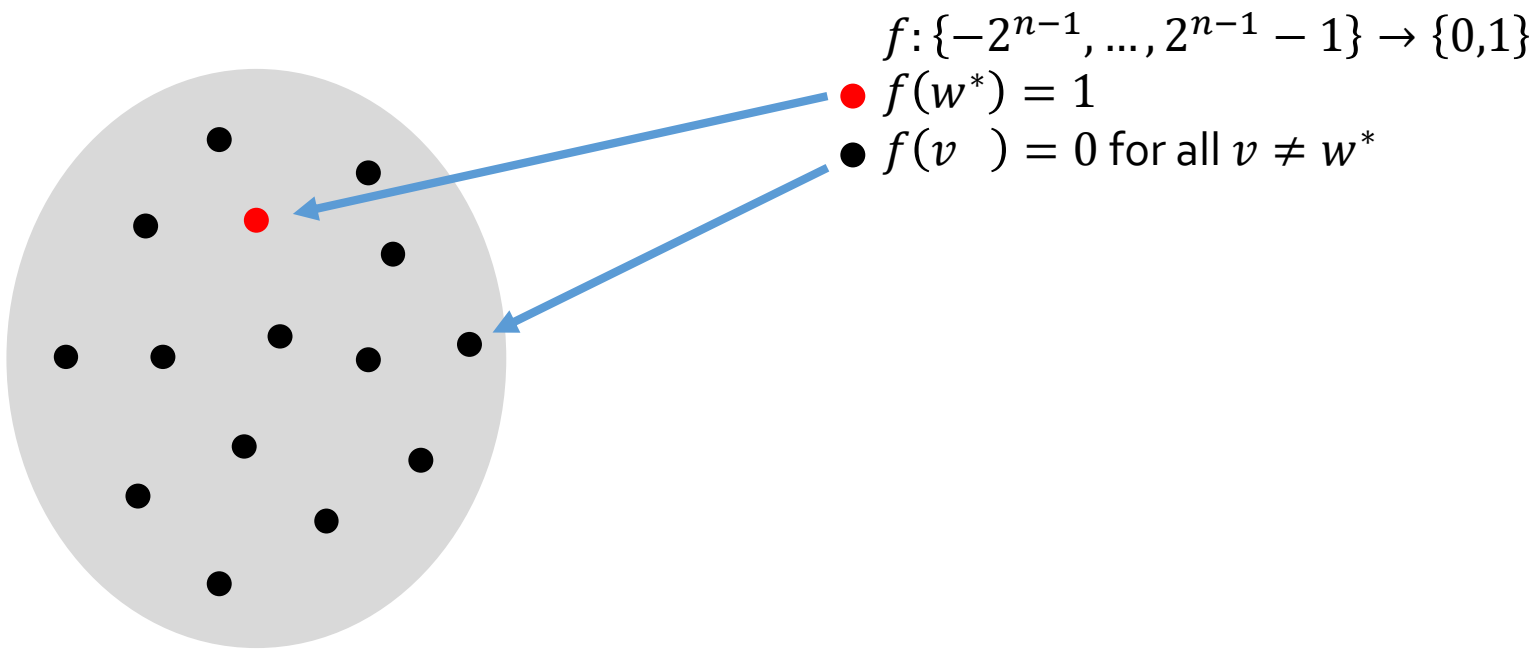

```
x:=H(x);
```




```
if x {     // controlled on x,
  y:=H(y); // apply H to y
}
```



Grover's Algorithm - Recap

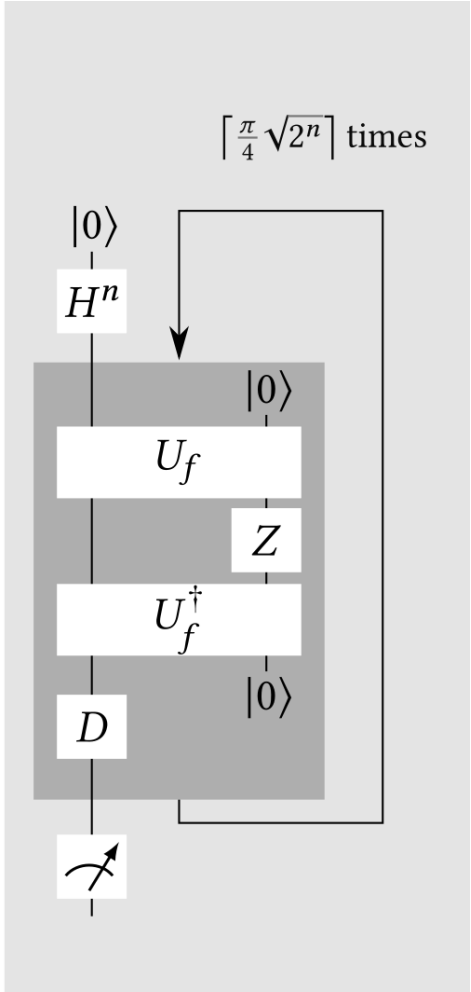


Grover's Algorithm in Silq

```
def grover[n:!ℕ](f:const int[n]!qfree→ 𝔹){
    nIterations:=⌈π/4 √(2^n)⌉;
    cand:=0:int[n];
    for k in [0..n) { cand[k] := H(cand[k]); }
    for k in [0..nIterations){

      if f(cand) { phase(π); }


      cand:=groverDiff[n](cand);
    }
    return measure(cand);
}
```

Grover's Algorithm in Silq

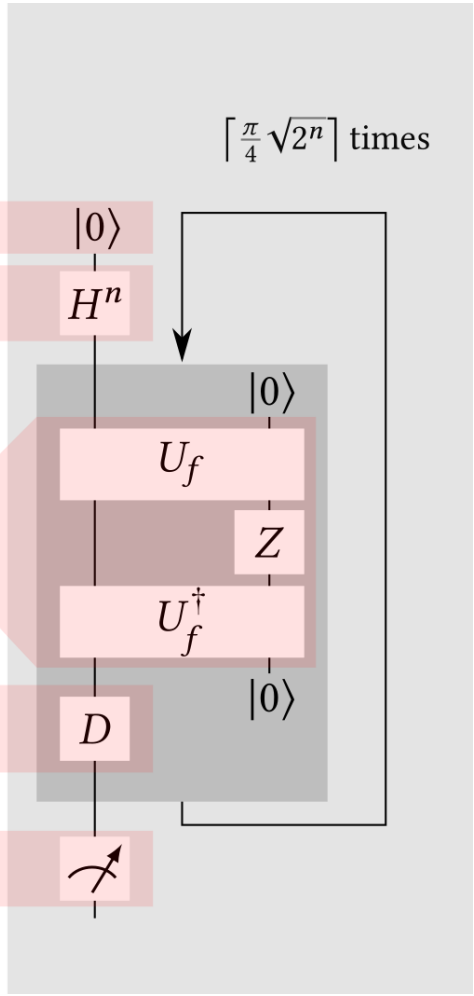
```
def grover[n:!ℕ](f:const int[n]!qfree→ 𝔹){
    nIterations:=⌈ π/4 √(2^n) ⌉;
    cand:=0:int[n];
    for k in [0..n) { cand[k] := H(cand[k]); }
    for k in [0..nIterations){

      if f(cand) { phase(π); }


      cand:=groverDiff[n](cand);
    }
    return measure(cand);
}
```



Time

Grover's Algorithm in Silq

```
def grover[n:!ℕ](f:const int[n]!qfree→ 𝔹){
    nIterations:=⌈ π/4 √(2^n) ⌉;
    cand:=0:int[n];
    for k in [0..n) { cand[k] := H(cand[k]); }
    for k in [0..nIterations){

      if f(cand) { phase(π); }


      cand:=groverDiff[n](cand);
    }
    return measure(cand);
}
```



Grover's Algorithm in Silq

```
1  def grover[n:!ℕ](f:const int[n]!qfree→ 𝔹){
```

Grover's Algorithm in Silq

generic
parameter n

```
1  def grover[n:!ℕ](f:const int[n]!qfree⟶ 𝔹){
```

Grover's Algorithm in Silq

generic parameter n

```
1  def grover[n:!ℕ](f:const int[n]!--qfree-->𝔹){
```

! : classical

ℕ : natural number

Type	Classical or Quantum?
B	Classical / Quantum
int[n] / uint[n]	Classical / Quantum
ℕ, ℤ, ℚ, ℝ	Classical
Generic parameters	Classical

Grover's Algorithm in Silq

generic parameter n

n-bit int

```
1  def grover[n:!ℕ](f:const int[n]!qfree→ 𝔹){
```

! : classical

ℕ : natural number

Grover's Algorithm in Silq

generic parameter n

n-bit int

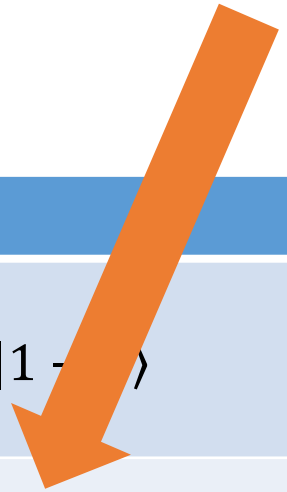
```
1  def grover[n:!ℕ](f:const int[n]!--qfree-->ℬ){
```

! : classical

ℕ : natural number

function consists of classical operations

g classical bijection



Function	Semantics	Qfree?
X	$\sum_{v=0}^{1} \gamma_v \lvert v\rangle \mapsto \sum_{v=0}^{1} \gamma_v \lvert 1 - v\rangle$	Qfree
General (non-classical)	$\sum_{\vec{v}} \gamma_{\vec{v}} \lvert \vec{v}\rangle \mapsto \sum_{\vec{v}} \gamma_{\vec{v}} \lvert g(\vec{v})\rangle$	Qfree
H	$\sum_{v=0}^{1} \gamma_v \lvert v\rangle \mapsto \sum_{v=0}^{1} \gamma_v (\lvert 0\rangle + (-1)^v \lvert 1\rangle)$	Not Qfree

Grover's Algorithm in Silq

generic parameter n

f preserves argument

n-bit int

```
1  def grover[n:!ℕ](f:const int[n]!--qfree-->ℙ){
```

! : classical

ℕ : natural number

function consists of classical operations

y:=f(x);

$$\sum_v \gamma_v |\varphi_v\rangle|v\rangle_x \mapsto \sum_v \gamma_v |\varphi_v\rangle|v\rangle_x|f(v)\rangle_y$$

qfree with $g(v) = (v, f(v))$

Grover's Algorithm in Silq

generic parameter n

f preserves argument

n-bit int

```
1  def grover[n:!ℕ](f:const int[n]!qfree⟶ 𝔹){
```

! : classical

ℕ : natural number

function consists of classical operations

$$\begin{aligned}
\textbf{H}&: \mathbb{B}\,! \xrightarrow{\text{mfree}} \mathbb{B}\\
\textbf{phase}&: !\mathbb{R}\,! \xrightarrow{\text{mfree}} \mathbb{1}\\
\textbf{measure}&: \tau\,! \longrightarrow !\tau\\
\texttt{groverDiff[n]}&: \textbf{int}[n]\,! \xrightarrow{\text{mfree}} \textbf{int}[n]
\end{aligned}$$

Grover's Algorithm in Silq

generic parameter n

f preserves argument

n-bit int

```
1  def grover[n:!ℕ](f:const int[n]!qfree→ 𝔹){
```

! : classical

ℕ : natural number

function consists of classical operations

```
2      nIterations:=⌈(π/4)√(2^n)⌉;
3      cand:=0:int[n];
4      for k in [0..n) { cand[k] := H(cand[k]); }
5
6      for k in [0..nIterations){
7          if f(cand) { phase(π); }
8          cand:=groverDiff[n](cand);
9      }
10     return measure(cand);
11 }
```

$$\psi_1 = |f\rangle_{\mathsf{f}} \otimes |n\rangle_{\mathsf{n}}$$

$$\psi_2 = \psi_1 \otimes \left|\left\lceil \frac{\pi}{4}\sqrt{2^n} \right\rceil\right\rangle_{\mathsf{nIterations}}$$

$$\begin{aligned}
\mathbf{H}: &\quad \mathbb{B}\,! \xrightarrow{\text{mfree}} \mathbb{B} \\
\mathbf{phase}: &\quad !\mathbb{R}\,! \xrightarrow{\text{mfree}} \mathbb{1} \\
\mathbf{measure}: &\quad \tau\,! \longrightarrow !\tau \\
\mathtt{groverDiff[n]}: &\quad \mathbf{int}[\mathsf{n}]\,! \xrightarrow{\text{mfree}} \mathbf{int}[\mathsf{n}]
\end{aligned}$$

Automatic Uncomputation

```
if f(cand) {
  phase(π);
}
```

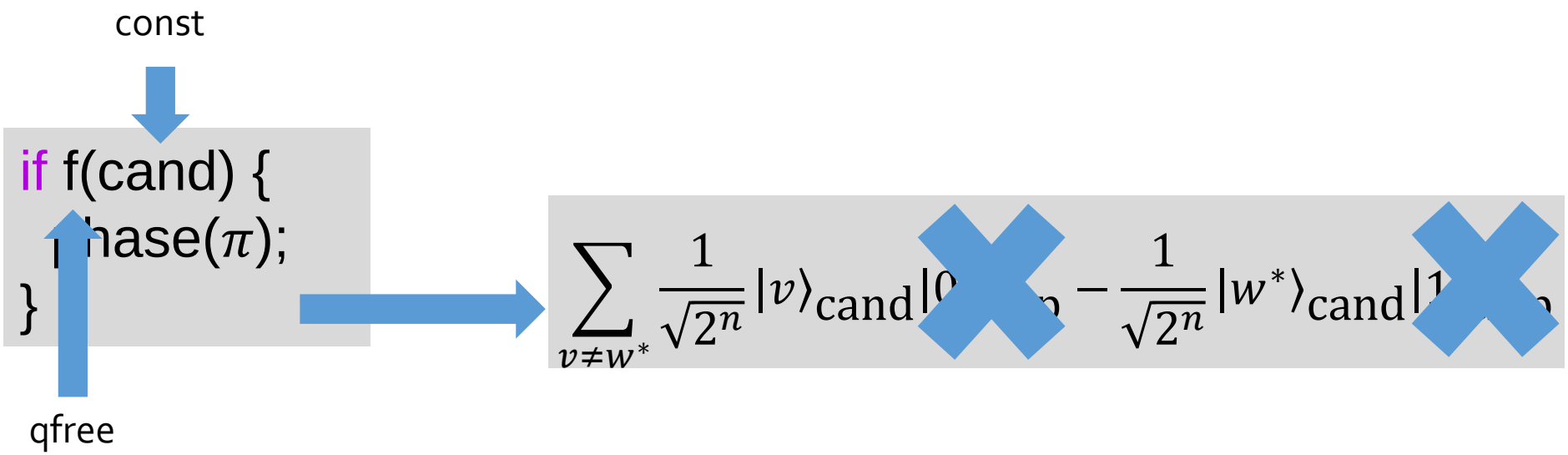
$$\sum_{v \neq w^*} \frac{1}{\sqrt{2^n}} |v\rangle_{\text{cand}} |0\rangle_{\text{tmp}} - \frac{1}{\sqrt{2^n}} |w^*\rangle_{\text{cand}} |1\rangle_{\text{tmp}}$$

implicit measurement

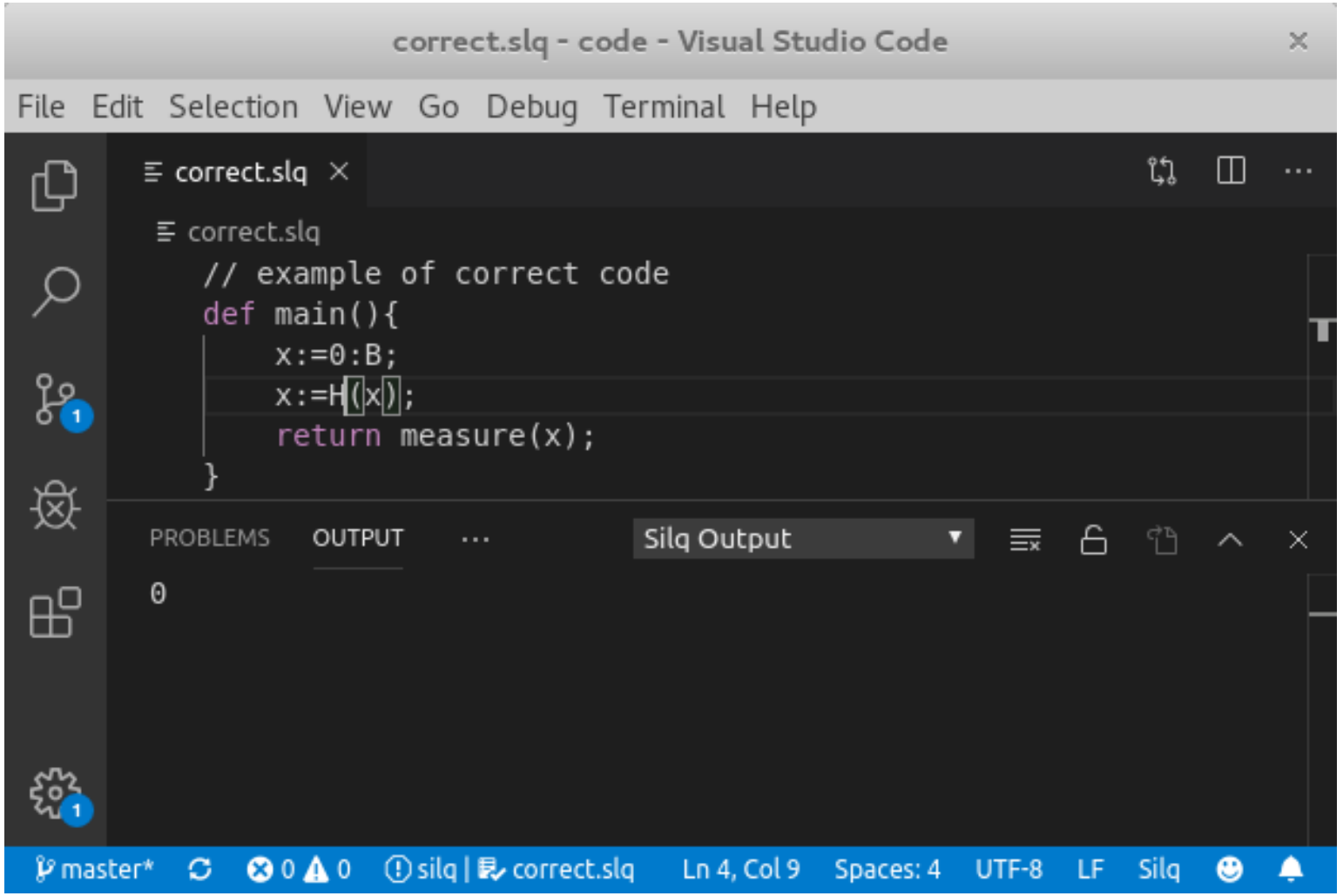
$$\sum_{v \neq w^*} \frac{1}{\sqrt{2^n}} |v\rangle_{\text{cand}} |0\rangle_{\text{tmp}}$$

$$\frac{1}{\sqrt{2^n}} |w^*\rangle_{\text{cand}} |1\rangle_{\text{tmp}}$$

Automatic Uncomputation

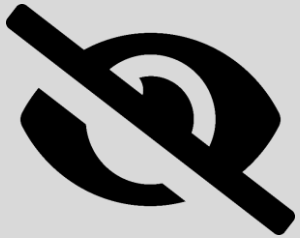


Live Coding

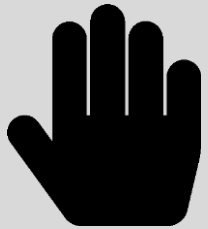

correct.slq - code - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
correct.slq
correct.slq
// example of correct code
def main(){
x:=0:B;
x:=H(x);
return measure(x);
}
PROBLEMS
OUTPUT
Silq Output
0
master*
0 0
silq | correct.slq
Ln 4, Col 9
Spaces: 4
UTF-8
LF
Silq

Summary of Type Annotations

Documentation on http://silq.ethz.ch/documentation#/documentation/1_annotations



mfree



const

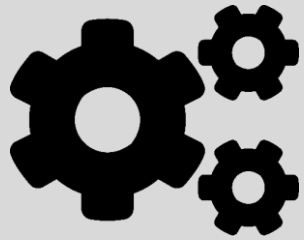
classical

qfree

Source of icons: https://fontawesome.com

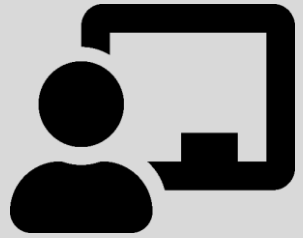
Downstream Applications

Simulation



Compilation

Research



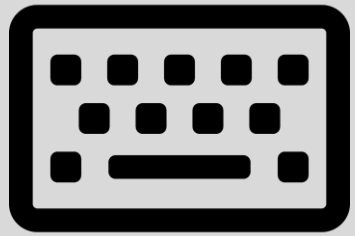
Teaching

Source of icons: https://fontawesome.com

Try Silq Yourself!

Install

Follow instructions on
http://www.silq.ethz.ch/install

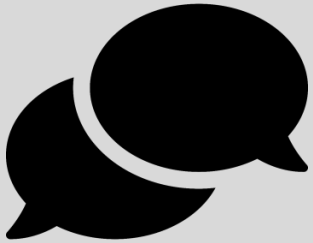


Try examples

Try to solve tasks / run solutions from
http://www.silq.ethz.ch/examples

Source of icons: https://fontawesome.com

Interested in Silq?

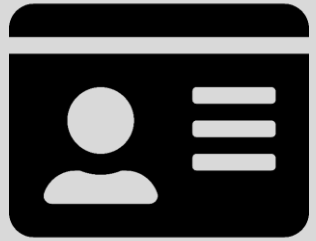


Ask &
Complain!

In person or
via Slack channel

Contact us!

Bachelor/Master thesis
Research projects
PhD



Benjamin Bichsel
Max Baader
Timon Gehr
Prof. Martin Vechev
(ETH Zürich)

https://www.sri.inf.ethz.ch/

Source of icons: https://fontawesome.com

Invalid Programs From Live Coding

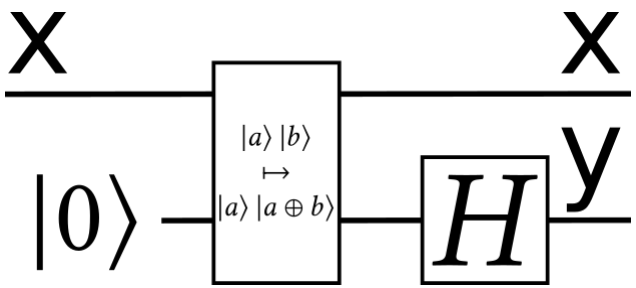
- The following slides are here for completeness.

Preventing Errors- Use Consumed

```
def useConsumed(x:𝔹){
  y := H(x);
  return (x,y); // undefined identifier x
}
```



```
def duplicateConst(const x:𝔹){
  y := H(x);
  return (x,y); // no error
}
```



Preventing Errors- Implicit Measurements

```
def implicitMeas[n:!ℕ](x:uint[n]){
  y := x % 2;
  return y;
} // parameter 'x' is not consumed
```

```
def unconsumedConst[n:!ℕ](const x:uint[n]){
  y := x % 2;
  return y;
} // no error
```

Preventing Errors- Conditional Measurements

```
def condMeas(const c:𝔹,x:𝔹){
  if c{
    x:= measure(x);
  }
  return x;
} // cannot call function 'measure[𝔹]' in 'mfree' context
```

```
def classCondMeas(const c:!𝔹,x:𝔹){
  if c{
    x:= measure(x):𝔹;
  }
  return x;
} // no error
```

Preventing Errors- Reverse Measurements

```
def revMeas(){
  return reverse(measure);
} // reversed function must be mfree
```

Preventing Errors- Invalid Uncomputation

```
def nonConst(y:𝔹){
  if X(y) { // X consumes y
    phase(π);
  }
} // non-'lifted' quantum expression must be consumed
```

```
def signFlipOf0(const y:𝔹){
  if X(y) { // X consumes a copy of y
    phase(π);
  }
} // no error
```

Preventing Errors- Invalid Uncomputation

```
def nonQfree(const y:𝔹,z:𝔹){
  if H(y) {
    z := X(z);
  }
  return z;
} // non-'lifted' quantum expression must be consumed
```