Quantum Computing

Lecture $|14\rangle$: **An Introduction to Silq**

Paolo Zuliani

Dipartimento di Informatica

Università di Roma "La Sapienza", Rome, Italy

What is Silq?

Silq is a new (2020) quantum programming language for implementing quantum algorithms as “programs” rather than “circuits”.

`https://silq.ethz.ch`

Silq: Key Features

- Silq can **mix quantum** and **classical code** (the computing model is akin to a classical computer driving a quantum coprocessor).
- Silq has a **type system** that goes beyond data types, based on code annotations. If your code 'type-checks', then it is a valid quantum transformation.
- Silq offers **automatic uncomputation** of subroutines to prevent side effects caused by measurement of quantum variables when they leave scope.

Silq Types

Basic types:

B (booleans), N (naturals), Q (rationals), `int[n]`, `uint[n]` (n-bit signed & unsigned integers), and more ...

Constructor types:

$s_1 \times \cdots \times s_n$	Cartesian product of types $s_1, \ldots, s_n$
$s[\,]$	list of type s
$s\hat{}n$	n-vector of type s
$!s$	classical type (**cannot** be in superposition)
$s \rightarrow t$	function that maps an object of type t to one of type t

The measurement operator has type $t \rightarrow !t$, where t is a quantum type.

Silq Types

Careful:

- by default B and `int[n]`, `uint[n]` are **quantum types**!
- N and Q can only be used with ! (i.e., they **must be** classical types)
- the type of a **classical function** that maps t to t' is written $t! \to t'$

Silq Annotations

`mfree` function: does **not measure** (any part of) its input.

Examples: any classical code, H (Hadamard), X (NOT), etc.

It gets type $s \rightarrow \text{mfree}\, t$

`qfree` function: does **neither introduce nor destroy superpositions** in the input.

A classical function that can be applied to a quantum input (an oracle!)

H is **not** `qfree`, but X is.

It gets type $s \rightarrow \text{qfree}\, t$.

Any `qfree` function is (obviously) `mfree`.

Silq Annotations

`const` parameter:	the callee function does **not** modify the parameter. Essentially, `const` parameters are used as read-only controls. Any other parameter will **not** be accessible after the execution of the function, which **consumes** the parameter (no cloning!)
`lifted` function:	`qfree` function with exclusively `const` parameters.

Generic Parameters

Silq allows defining functions with **classical** parameters that are known at **compile time**. Generic parameters are given in **square brackets**.

```
def tsquared[n:!N](a:!N^n) qfree {
  for i in [0..n){
    a[i] = a[i]^2;
  }
  return a
}
```

We can call `tsquared(2, 3)`, `tsquared(0, 34, 4037, 49)`, etc.

More on `const` Parameters

If a parameter is not `const`, then the function is supposed to **consume** it.

Remember that by default `uint[n]` is a quantum type!

```
def discard[n:!N](x:uint[n]) {
  y := x % 2;
  return y;
}
```

The function does not consume `x`, so that would constitute a **'silent' discard** of `x`: the type system of Silq hence rejects the code of the function.

Declaring `x` as `const` fixes the problem.

More on `const` Parameters

To handle `const` parameters, Silq first **duplicates** them, then consumes the duplicate.

$$\sum_x \alpha_x |x\rangle \longrightarrow \sum_x \alpha_x |x\rangle |x\rangle$$

Duplication is a unitary transformation and is **not** cloning (which is impossible!)

Example: duplicating a single qubit

$$\alpha |0\rangle + \beta |1\rangle \longrightarrow \alpha |00\rangle + \beta |11\rangle$$

can be achieved using a single $CNOT$.

The Deutsch-Jozsa Algorithm (I)

```
def bitwise_map [n:!N] (bits:B^n, f:B! -> B) {
  for i in [0..n) {
    bits[i]:=f(bits[i]);
  }
  return bits;
}
```

The Deutsch-Jozsa Algorithm (II)

```
def DJ[n:!N](const f:B^n! -> B^n) {
  state:=(0:int[n]) as B^n;
  state[n]:= X(state[n]);          // prepare state
  state:=bitwise_map(state, H);    // apply Hadamards
  state:=f(state);                 // apply oracle
  state:=bitwise_map(state, H);    // apply Hadamards
  state[n]:=H(state[n]);
  // return false is f is constant and true if f balanced
  return measure(state) == ((0:int[n]) as B^n);
}
```

Grover's Algorithm

generic parameter n — f preserves argument — n-bit uint — boolean

```
def grover[n:!ℕ](f:const uint[n]!─qfree→ 𝔹){
  nIterations:=⌊π/4 /asin(2^(−n/2))⌋;
  cand:=0:uint[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);
}
```

! : classical
ℕ : natural number

f preserves superpositions

variable n holds value n

$$\mathbf{H}: \quad \mathbb{B}\,! \xrightarrow{\texttt{mfree}} \mathbb{B}$$

$$\texttt{groverDiff[n]}: \mathbf{uint}[n]\,! \xrightarrow{\texttt{mfree}} \mathbf{uint}[n]$$

$$\mathbf{measure}: \quad \tau\,! \longrightarrow !\tau$$

$$\mathbf{phase}: \; !\mathbf{float}\,! \xrightarrow{\texttt{mfree}} \mathbb{1}$$

unit type

$$\psi_1 = \left|\tilde{f}\right\rangle_{\mathtt{f}} \otimes |n\rangle_{\mathtt{n}}$$

$$\psi_2 = \psi_1 \otimes \left|\left\lfloor \frac{\pi}{4} / \operatorname{asin}\left(2^{-n/2}\right)\right\rfloor\right\rangle_{\mathtt{nIterations}}$$

$$\psi_4 = \psi_2 \otimes \sum_{v} \frac{1}{\sqrt{2^n}} |v\rangle_{\mathtt{cand}}$$

$$\psi_6^{(0)} = \psi_2 \otimes |0\rangle_{\mathtt{k}} \otimes \left(\sum_{v \neq w^\star} \frac{1}{\sqrt{2^n}} |v\rangle_{\mathtt{cand}} + \frac{1}{\sqrt{2^n}} |w^\star\rangle_{\mathtt{cand}} \right)$$

then branch

$$\psi_7^{(0)} = \psi_2 \otimes |0\rangle_{\mathtt{k}} \otimes \left(\sum_{v \neq w^\star} \frac{1}{\sqrt{2^n}} |v\rangle_{\mathtt{cand}} |0\rangle_{\underline{\mathtt{f(cand)}}} + \frac{1}{\sqrt{2^n}} |w^\star\rangle_{\mathtt{cand}} |1\rangle_{\underline{\mathtt{f(cand)}}} \right)$$

$$\psi_8^{(0)} = \psi_2 \otimes |0\rangle_{\mathtt{k}} \otimes \left(\sum_{v \neq w^\star} \frac{1}{\sqrt{2^n}} |v\rangle_{\mathtt{cand}} |0\rangle_{\underline{\mathtt{f(cand)}}} + -\frac{1}{\sqrt{2^n}} |w^\star\rangle_{\mathtt{cand}} |1\rangle_{\underline{\mathtt{f(cand)}}} \right)$$

$$\psi_9^{(0)} = \psi_2 \otimes |0\rangle_{\mathtt{k}} \otimes \left(\sum_{v \neq w^\star} \frac{1}{\sqrt{2^n}} |v\rangle_{\mathtt{cand}} + -\frac{1}{\sqrt{2^n}} |w^\star\rangle_{\mathtt{cand}} \right)$$

$$\psi_{10}^{(0)} = \psi_2 \otimes |0\rangle_{\mathtt{k}} \otimes \left(\sum_{v \neq w^\star} \gamma_v^- |v\rangle_{\mathtt{cand}} + \gamma_{w^\star}^+ |w^\star\rangle_{\mathtt{cand}} \right)$$