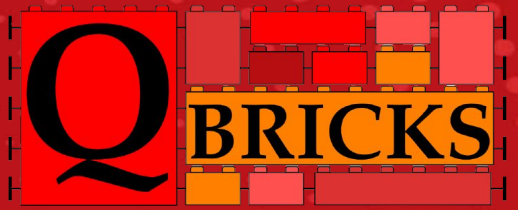
DE LA RECHERCHE À L'INDUSTRIE

Qbricks: specification, development and verification of quantum programs



Christophe Chareton, Sébastien Bardin

IQFA 'XII 12th Colloquium on Quantum Engineering, Fundamental Aspects to Applications

Take away

- Quantum computers arrive

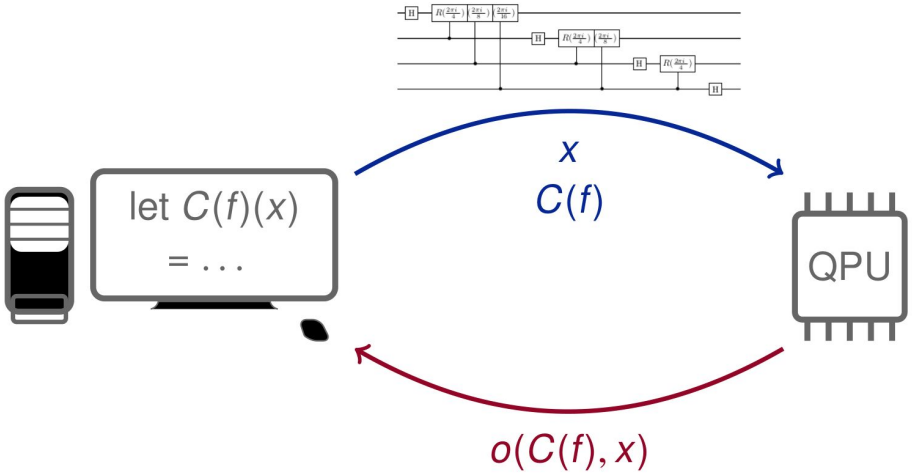
  →How to write **correct** programs?
- Need specification and verification mechanisms
  - parametrized
  - largely automated
- We are developing Qbricks as a first step towards this goal
  - core building circuit language
  - circuit semantics
  - high level specification framework
  - proof engine
- Certified parametrized implementation of the **Shor order finding** algorithm (Shor-OF)

The hybrid model

A quantum co-processor (QPU), controlled by a classical computer

- classical control flow
- quantum computing requests, sent to the QPU

→structured sequenced of instructions: **quantum circuits**



Algorithm specifications

Algorithm: Quantum order-finding

Inputs: (1) A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \rightarrow |j\rangle|x^j k \bmod N\rangle$, for x co-prime to the L-bit number N, (2) $t = 2L + 1 + \lceil \log\left(2 + \frac{1}{2\epsilon}\right) \rceil$ qubits initialized to $|0\rangle$, and (3) L qubits initialized to the state $|1\rangle$.

Outputs: The least integer $r > 0$ such that $x^r = 1 \pmod N$.

Runtime: $O(L^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. $|0\rangle|1\rangle$ — initial state
2. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ — create superposition
3. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$ — apply $U_{x,N}$

   $\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j\rangle|u_s\rangle$
4. $\rightarrow \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\widetilde{s/r}\rangle|u_s\rangle$ — apply inverse Fourier transform to first register
5. $\rightarrow \widetilde{s/r}$ — measure first register
6. $\rightarrow r$ — apply continued fractions algorithm

Shor-OF (from N & C, p. 232)

A specification preamble:

- **Input parameters (size, oracle, etc)**
- **Functional correctness**: Inputs-Outputs relation
- **Complexity**: number of elementary operations

Validation of quantum programs

Algorithm: Quantum order-finding

Inputs: (1) A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \to |j\rangle|x^j k \bmod N\rangle$, for x co-prime to the L-bit number N, (2) $t = 2L + 1 + \lceil \log\,(2 + \frac{1}{2\varepsilon}) \rceil$ qubits initialized to $|0\rangle$, and (3) L qubits to the state $|1\rangle$.

Outputs: The least integer $r > 0$ such that $x^r = 1 \pmod N$.

Runtime: $O(L^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

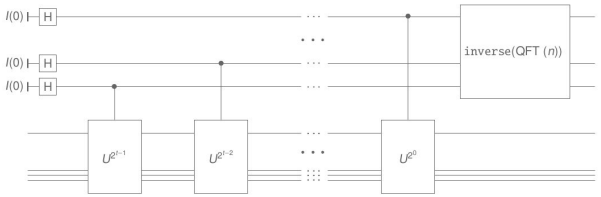
1. $|0\rangle|1\rangle$ — initial state
2. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ — create superposition
3. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$ — apply $U_{x,N}$

   $\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j\rangle|u_s\rangle$
4. $\to \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\widetilde{s/r}\rangle|u_s\rangle$ — apply inverse Fourier tran register
5. $\to \widetilde{s/r}$ — measure first register
6. $\to r$ — apply continued fractions algorithm

```
qft_internal :: [Qubit] -> Circ [Qubit]
qft_internal [] = return []
qft_internal [x] = do
  hadamard x
  return [x]
qft_internal (x:xs) = do
  xs' <- qft_internal xs
  xs'' <- rotations x xs' (length xs')
  x' <- hadamard x
  return (x':xs'')
  where
    -- Auxiliary function used by 'qft'.
    rotations :: Qubit -> [Qubit] -> Int -> Circ [Qubit]
    rotations _ [] _ = return []
    rotations c (q:qs) n = do
      qs' <- rotations c qs n
      q' <- rGate ((n + 1) - length qs) q `controlled` c
      return (q':qs')
```

A specification preamble:

- **Input parameters (size, oracle, etc)**
- **Functional correctness**: Inputs-Outputs relation
- **Complexity**: number of elementary operations



Validation of quantum programs

Algorithm: Quantum order-finding

Inputs: (1) A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \to |j\rangle|x^j k \bmod N\rangle$, for x co-prime to the L-bit number N, (2 $t = 2L + 1 + \lceil \log\,(2 + \frac{1}{2\varepsilon}) \rceil$ qubits initialized to $|0\rangle$, and (3) L qubits to the state $|1\rangle$.

Outputs: The least integer $r > 0$ such that $x^r = 1 \pmod N$.

Runtime: $O(L^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. $|0\rangle|1\rangle$ — initial state
2. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ — create superposition
3. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$ — apply $U_{x,N}$

   $\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j\rangle|u_s\rangle$
4. $\to \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\widetilde{s/r}\rangle|u_s\rangle$ — apply inverse Fourier tran register
5. $\to \widetilde{s/r}$ — measure first register
6. $\to r$ — apply continued fractions algorithm

```
q
q
q

q
```

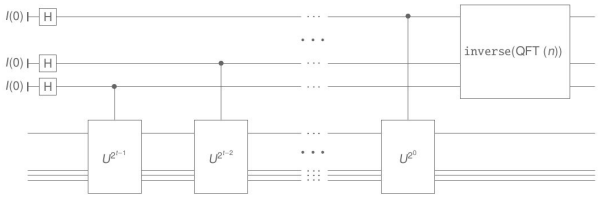
```
where
  -- Auxiliary function used by 'qft'.
  rotations :: Qubit -> [Qubit] -> Int -> Circ [Qubit]
  rotations _ [] _ = return []
  rotations c (q:qs) n = do
    qs' <- rotations c qs n
    q' <- rGate ((n + 1) - length qs) q `controlled` c
    return (q':qs')
```

Higher-order programming :

- Quantum program : Inputs → quantum circuits
- Quantum circuit : quantum data → quantum data

A specification preamble:

- **Input parameters (size, oracle, etc)**
- **Functional correctness**: Inputs-Outputs relation
- **Complexity**: number of elementary operations



Validation of quantum programs

Algorithm: Quantum order-finding

Inputs: (1) A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \to |j\rangle|x^j k \bmod N\rangle$, for x co-prime to the L-bit number N, (2) $t = 2L + 1 + \lceil \log\left(2 + \frac{1}{2\epsilon}\right) \rceil$ qubits initialized to $|0\rangle$, and (3) L qubits to the state $|1\rangle$.

Outputs: The least integer $r > 0$ such that $x^r = 1 \pmod N$.

Runtime: $O(L^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. $|0\rangle|1\rangle$ — initial state
2. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ — create superposition
3. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$ — apply $U_{x,N}$

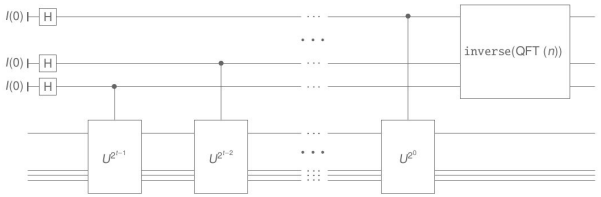
   $\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j\rangle|u_s\rangle$
4. $\to \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\widetilde{s/r}\rangle|u_s\rangle$ — apply inverse Fourier tran… register
5. $\to \widetilde{s/r}$ — measure first register
6. $\to r$ — apply continued fractions algorithm

```
qft_internal :: [Q
qft_internal [] =
qft_internal [x] =
  hadamard x
  return [x]
qft_internal (x:xs
  xs' <- qft_inter
  xs'' <- rotation
  x' <- hadamard x
  return (x':xs'')
  where
    -- Auxiliary function used by 'qft'.
    rotations :: Qubit -> [Qubit] -> Int -> Circ [Qubit]
    rotations _ [] _ = return []
    rotations c (q:qs) n = do
      qs' <- rotations c qs n
      q' <- rGate ((n + 1) - length qs) q `controlled` c
      return (q':qs')
```

- Specification: the circuit should meet the spec **for any value of parameters**
- Quantum programming is non-intuitive
  →High risk for bugs!

A specification preamble:

- **Input parameters (size, oracle, etc)**
- **Functional correctness**: Inputs-Outputs relation
- **Complexity**: number of elementary operations



Validation of quantum programs

Algorithm: Quantum order-finding

Inputs: (1) A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \to |j\rangle|x^j k \bmod N\rangle$, for x co-prime to the L-bit number N, (2) $t = 2L + 1 + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ qubits initialized to $|0\rangle$, and (3) L qubits ini to the state $|1\rangle$.

Outputs: The least integer $r > 0$ such that $x^r = 1 \pmod N$.

Runtime: $O(L^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. $|0\rangle|1\rangle$ — initial state
2. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ — create superposition
3. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$ — apply $U_{x,N}$

$$\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j\rangle|u_s\rangle$$

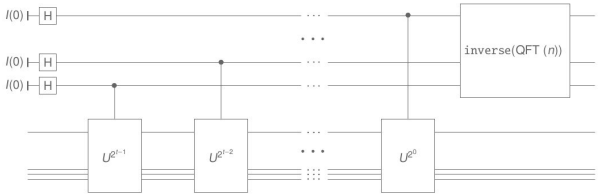
```
qft_internal :: [Q
qft_internal [] =
qft_internal [x] =
  hadamard x
  return [x]
qft_internal (x:xs
  xs' <- qft_inter
  xs'' <- rotation
  x' <- hadamard x
  return (x':xs'')
  where
    -- Auxiliary function used by 'qft'.
    rotations :: Qubit -> [Qubit] -> Int -> Circ [Qubit]
    rotations _ [] _ = return []
    c (q:qs) n = do
    otations c qs n
    ate ((n + 1) - length qs) q `controlled` c
    ':qs')
```

- Specification: the circuit should meet the spec **for any value of parameters**
- Quantum programming is non-intuitive
  →High risk for bugs!

What about testing/assertion checking?

- **Destructive measurement** : No means to control an execution
- **Tests are expensive and often statistical**

- **Complexity**: number of elementary operations



Validation of quantum programs

Algorithm: Quantum order-finding

Inputs: (1) A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \to |j\rangle|x^j k \bmod N\rangle$, for x co-prime to the L-bit number N, (2) $t = 2L + 1 + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ qubits initialized to $|0\rangle$, and (3) L qubits ini to the state $|1\rangle$.

Outputs: The least integer $r > 0$ such that $x^r = 1 \pmod N$.

Runtime: $O(L^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. $|0\rangle|1\rangle$ — initial state
2. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ — create superposition
3. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$ — apply $U_{x,N}$

$$\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j\rangle|u_s\rangle$$

```
qft_internal :: [Q
qft_internal [] =
qft_internal [x] =
  hadamard x
  return [x]
qft_internal (x:xs
  xs' <- qft_inter
  xs'' <- rotation
  x' <- hadamard x
  return (x':xs'')
  where
    -- Auxiliary function used by 'qft'.
    rotations :: Qubit -> [Qubit] -> Int -> Circ [Qubit]
    rotations _ [] _ = return []
    c (q:qs) n = do
    otations c qs n
    ate ((n + 1) -
    ':qs')
```

- Specification: the circuit should meet the spec **for any value of parameters**
- Quantum programming is non-intuitive
  →High risk for bugs!

What about testing/assertion checking?

- **Destructive measurement** : No means to control an execution
- **Tests are expensive and often statistical**

What about full **formal** verification?

- No need to execute
- unbounded state space
- absolute guarantee

- **Complexity**: number of elementary operations

QBRICKS / Description

- **State of the art:**

	Circuit	Parametrized	Proof automation	Complexity specifications
Path-sums (Univ. of Waterloo)	✓	✗	✓	✗
SQIR(Univ. of Maryland)	✓	✓	✗	could
QHL(Tsinghua univ.)	✗	✓	✗	✗
Trade-off automation Vs parametricity ? (higher-order reasoning)				
QBRICKS	✓	✓	↗	✓

Collaboration with Benoît Valiron

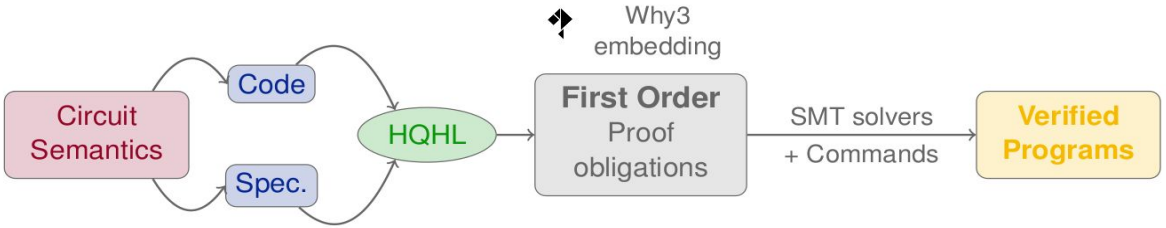
- **Main characteristics**

  - **Programming and validation** (functional correctness + complexity) **environment for quantum programs**
  - Equipped with,
    - **parametricity**, higher-order programming and **scale invariance**
    - **proof automation** via first order reasoning techniques

QBRICKS contributions

- A programming, Spec and proof framework, Qbricks-DSL + Qbricks-Spec + HQHL, yielding **first-order proof obligations**
- A flexible symbolic representation for **first order reasoning** about quantum states and transformations : **Parametrized Path Sums (PPS)**
- Dedicated mathematical libraries, +14 kLoC
- Non trivial case studies (Shor-OF, Grover, QPE, etc).

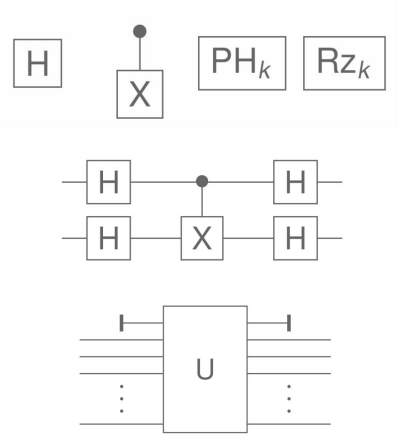
Publicly available at `https://qbricks.github.io/`



Programming framework: QBRICKS-DSL

- **Functional** programming + Why3 embedding
- Circuit as objects ⟶ no cloning **by construction**
- **Deductive verification** ⟶ use of **contracts** (well formedness + functional correctness + complexity)
- Unitary programs

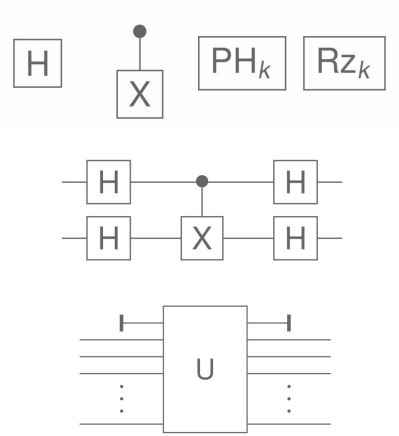
- A minimal set of primitive functions
  - elementary gates
  - compositions : parallel/sequence
  - ancilla creation/anihilation



- derived high-level combinators: inversion, control, qbit permutations, etc
- **bounded** iterations [essential trick]

Verification

- Focus now on the formal verification part
  - annotate code with contracts
  - compilation → “proof obligations”
  - PO proven → the program meets the contracts for all its possible input



Annotated code and deductive verif.

- provide absolute guarantees
- automate proofs
- industrial successes
- verify widespread languages (C, Java, Ocaml, etc)

Three main ingredients:

- operational semantics
- specification language
- proof engine

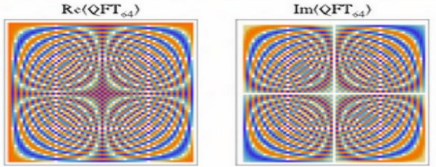
JML

frama C

Software Analyzers

Symbolic representation: path-sums

Matrix semantics: QFT(64):



$|0\rangle|u\rangle$ initial state

$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ create superposition

$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$ apply $U_{x,N}$

$\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} |j\rangle|u_s\rangle$

$\rightarrow \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\widetilde{s/r}\rangle|u_s\rangle$ apply inverse Fourier transform to the first register

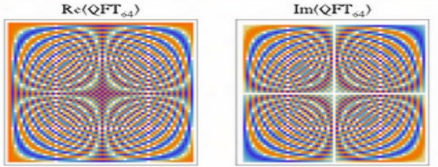
$\rightarrow |\widetilde{s/r}\rangle$ measure first register

$$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$$

$$\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} |j\rangle|u_s\rangle$$

Symbolic representation: path-sums

Matrix semantics: QFT(64):



$$|0\rangle|u\rangle \quad \text{initial state}$$
$$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle \quad \text{create superposition}$$
$$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle \quad \text{apply } U_{x,N}$$
$$\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} |j\rangle|u_s\rangle$$
$$\rightarrow \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \overline{|s/r\rangle}|u_s\rangle \quad \text{apply inverse Fourier transform to the first register}$$
$$\rightarrow |s/r\rangle \quad \text{measure first register}$$

$$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$$
$$\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} |j\rangle|u_s\rangle$$

Path-sum semantics[Amy 2019]: build language term

$$PS : |k\rangle \rightarrow \frac{1}{\sqrt{2^r}} \sum_{j=0}^{2^{r-1}} e^{2\cdot\pi\cdot f(k,j)} |g(k,j)\rangle$$

With instances of language variables:

- `r: int`
- `f: bitvec → bitvec → complex`
- `g: bitvec → bitvec → bitvec`

- Concise in practice
- Good compositional properties
- But no parametricity

Our extension: parametrized path-sums

Parametrized circuits $C(\vec{p})$:

$$\mathrm{PPS}_{C,\vec{p}} : |k\rangle \rightarrow \frac{1}{\sqrt{2^{r_{C,\vec{p}}}}} \sum_{j=0}^{2^{r_{C,\vec{p}}}-1} e^{2\cdot\pi\cdot f_{C,\vec{p}}(k,j)} |g_{C,\vec{p}}(k,j)\rangle$$

- simple first-order modular reasoning upon each component

$$r_{C,\vec{p}}, f_{C,\vec{p}}, g_{C,\vec{p}}$$

- simple first-order composition rules

→ at use for both spec (QBRicks-Spec) and deduction (HQHL)

Algorithm: Quantum phase estimation

Inputs: (1) A black box wich performs a controlled-U^j operation, for integer j, (2) an eigenstate $|u\rangle$ of U with eigenvalue $e^{2\pi i\varphi_u}$, and (3) $t = n + \lceil \log\left(2 + \frac{1}{2\epsilon}\right) \rceil$ qubits initialized to $|0\rangle$.

Outputs: An n-bit approximation $\widetilde{\varphi_u}$ to φ_u.

Runtime: $O(t^2)$ operations and one call to controlled-U^j black box. Succeeds with probability at least $1 - \epsilon$.

Procedure:

1.	$\|0\rangle\|u\rangle$	initial state
2.	$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} \|j\rangle\|u\rangle$	create superposition
3.	$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} \|j\rangle U^j\|u\rangle$	apply black box
	$= \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi i j\varphi_u}\|j\rangle\|u\rangle$	result of black box
4.	$\rightarrow \|\widetilde{\varphi_u}\rangle\|u\rangle$	apply inverse Fourier transform
5.	$\rightarrow \widetilde{\varphi_u}$	measure first register

Specification preamble

- input parameters + preconditions
- post-conditions:
  - functional
  - complexity

Body

- sequence of quantum operations,
- intermediate system state postconditions

QBRICKS / Demonstration

Decorated code

```
let  function apply_black_box (circ:circuit)(k n:int)
                              (ghost y:matrix complex)(ghost theta:complex)
```

```
= sequence (create_superposition k n) (black_box_circ k y theta)
```

Functional programming

`parameters -> quantum circuits`

QBRICKS / Demonstration

Decorated code

```
let  function apply_black_box (circ:circuit)(k n:int)
                              (ghost y:matrix complex)(ghost theta:complex)
  requires{n=k+width circ}
  requires{real_ theta}
  requires{0 < k < n}
  requires{is_a_ket_l y (n-k)}
  requires{eigen circ y (real_to_ang theta)}

  ensures{width result = n}
  ensures{ancillas result =0}
  ensures{size result<=n+k*size circ}

  ensures{path_sem result  (kron (ket k 0)y) =
          (kron (pow_inv_sqrt_2 k *..   ket_sum (n_bvs k)
           (fun x -> black_box_coeff  theta x *..  (bv_to_ket x)) k) y)}

 = sequence (create_superposition k n) (black_box circ k y theta)
```

Functional programming

`parameters -> quantum circuits`

Specifications

- preconditions
- complexity specifications
- functional assertions

QBRICKS / Demonstration

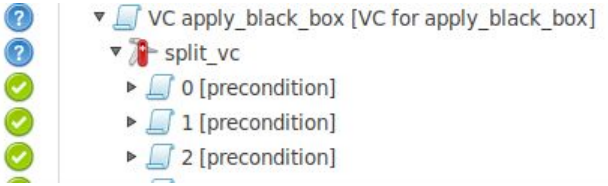
Proof obligations generation, via Why3 interface

```
let  function apply_black_box (circ:circuit)(k n:int)
                              (ghost y:matrix complex)(ghost theta:complex)
   requires{n=k+width circ}
   requires{real_ theta}
   requires{0 < k < n}
   requires{is_a_ket_l y (n-k)}
   requires{eigen circ y (real_to_ang theta)}

   ensures{width result = n}
   ensures{ancillas result =0}
   ensures{size result<=n+k*size circ}

   ensures{path_sem result  (kron (ket k 0)y) =
            (kron (pow_inv_sqrt_2 k *..   ket_sum (n_bvs k)
             (fun x -> black_box_coeff  theta x *..  (bv_to_ket x)) k) y)}

 = sequence (create_superposition k n) (black_box circ k y theta)
```

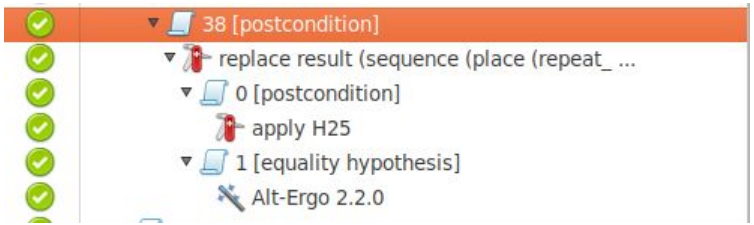


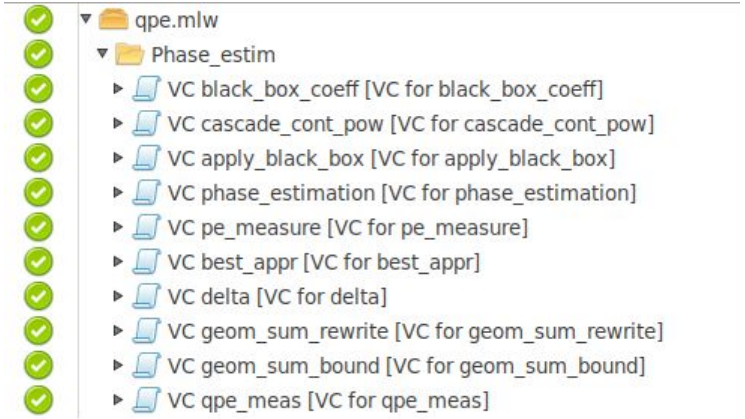

```
goal VC apply_black_box :
  path_sem result (kronecker (ket k 0) y)
  = kronecker
    (pow_inv_sqrt_2 k
     *.. ket_sum_l (n_bvs k)
         (fun (x:bitvec) -> black_box_coeff theta x *.. bv_to_ket x) k)
    y
```

QBRICKS / Demonstration

Proof support

- **Interfaces**
  - calls to SMT-solvers
  - interactive proof commands, to help SMT-solvers
- **Output**
  - probation against complex case studies (× 6 vs SotA)
  - high-level (95%) of automation, proof effort × 1/3 vs SotA





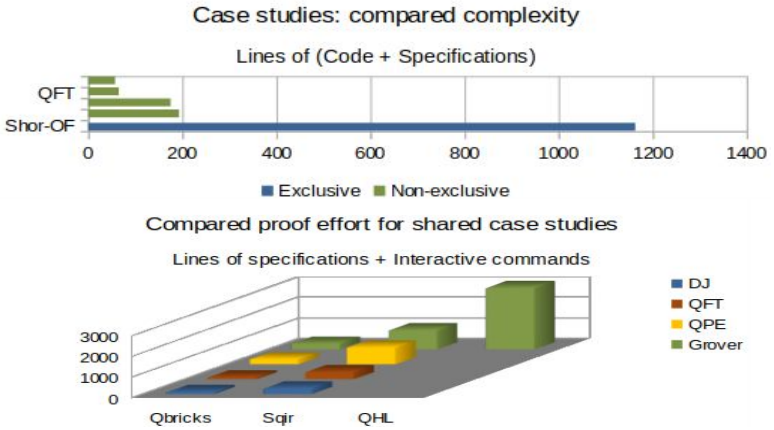
Dedicated mathematical libraries

	Lines of code	Lemmas	Modules	Definitions
Mathematics libraries	14695	1614	77	328
Sets	532	59	4	14
Algebra	2091	190	10	37
Arithmetics	538	77	4	7
Binary arithmetics	1778	189	8	42
Complex numbers	2226	344	15	57
Quantum data	3335	310	12	68
Exponentiation	843	100	4	4
Iterators	861	72	6	30
Functions	259	33	3	8
Kronecker product	420	41	2	8
Unity circle	1812	199	9	53

QBRICKS / Highlights

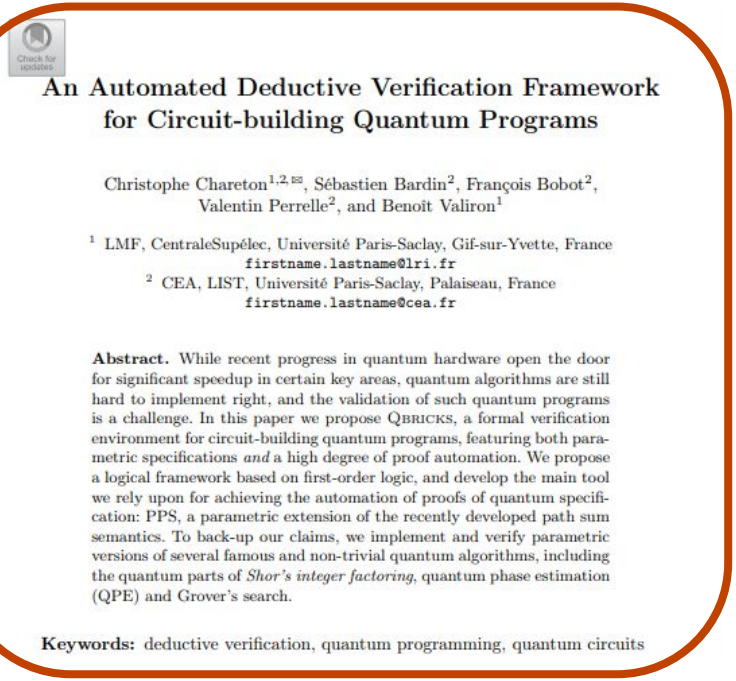
MAJOR ACHIEVEMENTS

- a core development framework for **parametrized verified programming**
- **first ever verified implementation of Shor order finding algorithm** (95% proof automation),



COMMUNICATION

- 1 publication in a **major conference** (ESOP 2021)
- 1 **book chapter** under review
- 6 talks is **specialized workshops**
- invitation to the 1st **Dagstuhl seminar** on quantum programs



An Automated Deductive Verification Framework for Circuit-building Quantum Programs

Christophe Chareton[1,2,✉], Sébastien Bardin[2], François Bobot[2], Valentin Perrelle[2], and Benoît Valiron[1]

[1] LMF, CentraleSupélec, Université Paris-Saclay, Gif-sur-Yvette, France
firstname.lastname@lri.fr
[2] CEA, LIST, Université Paris-Saclay, Palaiseau, France
firstname.lastname@cea.fr

Abstract. While recent progress in quantum hardware open the door for significant speedup in certain key areas, quantum algorithms are still hard to implement right, and the validation of such quantum programs is a challenge. In this paper we propose QBRICKS, a formal verification environment for circuit-building quantum programs, featuring both parametric specifications *and* a high degree of proof automation. We propose a logical framework based on first-order logic, and develop the main tool we rely upon for achieving the automation of proofs of quantum specification: PPS, a parametric extension of the recently developed path sum semantics. To back-up our claims, we implement and verify parametric versions of several famous and non-trivial quantum algorithms, including the quantum parts of *Shor's integer factoring*, quantum phase estimation (QPE) and Grover's search.

Keywords: deductive verification, quantum programming, quantum circuits

ROADMAP

- Verification system improvements
  - more automation
  - finer resource analysis tools
  - error sensitivity evaluation (NISQ)
- Formally verified circuit transformations
  - to/from alternative computing models (MBQC/ZX)
  - optimization
  - error correction
- Language design
  - integration in hybrid programming
  - interface with widespread programming languages
  - interface with further intuitive specification languages

Challenges for quantum program verification

- non standard theories (probabilities, complex numbers, kronecker product, etc)
- double layer programming paradigm: higher-order functions

  parametrized programming ⇔ automated proof support