On Quantum Programming Languages

Quantum λ-Calculus, Quipper and the Quantum Switch

Benoît Valiron, CentraleSupélec / LMF

Sept 24, 2024

Soutenance de HDR

Quantum Computation

Concept

- Quantum objects used for storing information
- Well-understood mathematical framework

At the time of my PhD defense in 2008

- Many quantum algorithms with theoretical speedup
- No concrete application forseen due to limited hardware
- Very preliminary quantum programming languages

Current state of the field

- Steady progresses in hardware
- Wide range of concrete use-cases
- Development of complex programming frameworks

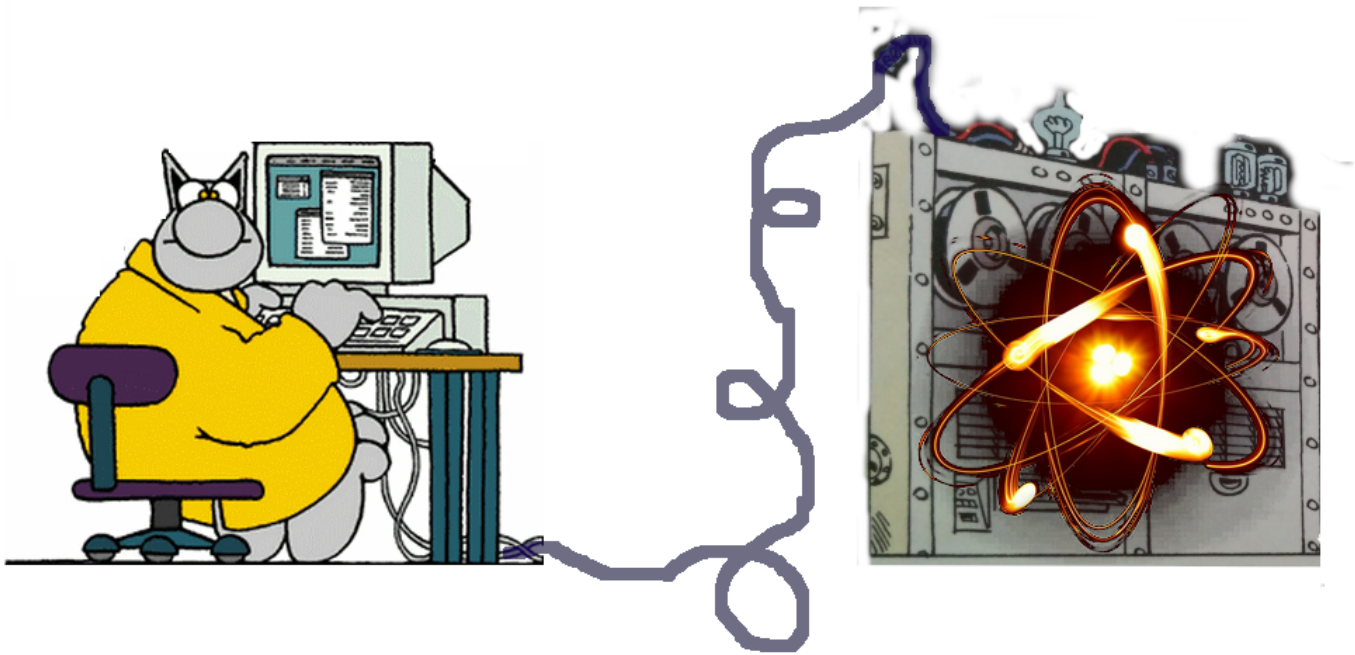
Standard Mathematical Representation

	Classical	Quantum
Basic register	bit	quantum bit = qbit
Math model	Set	Hilbert Spaces and Unitary maps
Pure states	$\mathcal{B} = \{\texttt{true}, \texttt{false}\}$	$\mathbb{C}^2 \equiv \langle \mathcal{B} \rangle \qquad \alpha \cdot \lvert 0 \rangle + \beta \cdot \lvert 1 \rangle$
3 registers	$\mathcal{B} \times \mathcal{B} \times \mathcal{B}$	$\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^8 \equiv \langle \mathcal{B}^3 \rangle$
With probability	Probability distributions	Density matrices and completely positive maps (CPM)

Entangled state Cannot be written a pair of states

$$\frac{1}{\sqrt{2}}(\lvert 00 \rangle + \lvert 11 \rangle) \quad \neq \quad \lvert \phi \rangle \otimes \lvert \psi \rangle$$

Standard Computational Model: Co-Processor



Main Computer	Interface	Co-processor
Program	Instructions	Quantum memory
Execution flow	⇄	Quantum operations
	Feedback	

Standard Computational Model: Co-Processor

The quantum memory

- A set of individually addressable quantum registers

Actions through the interface

- Initialize registers
- Apply quantum operations
  - Linear, unitary transformations on the state space
  - No cloning of quantum states
- Read register
  - Through probabilistic measurement
  - Only access to quantum information

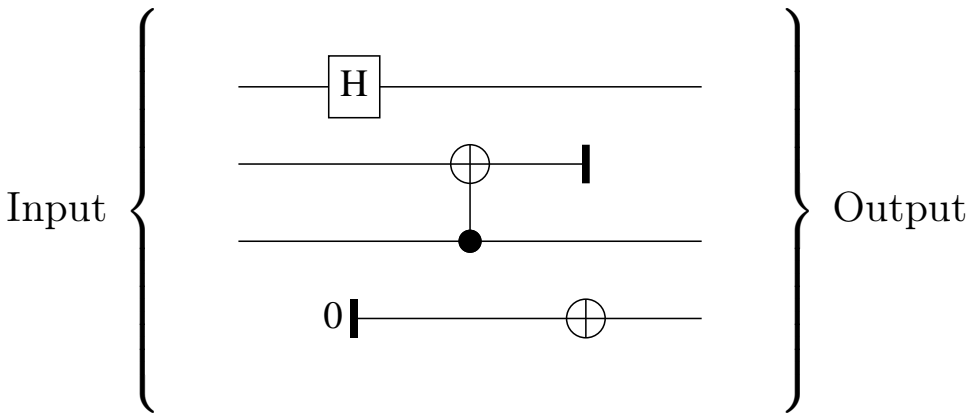
From the programmer's perspective

- A probabilistic read operation
- Non-duplicable data

Standard Computational Model: Co-Processor

Local Operations

- Batch of low-level instructions
- Elementary operations applied on the quantum memory
- Written as a quantum circuit




```
input 0 1 2
init 3
H 0
CNOT 2 1
NOT 3
discard 1
output 0 2 3
```

Limited control flow

- No loops or tests
- Rudimentary function calls through circuit composition

Challenge for Quantum Programming

Going from

- Low-level quantum datastructures
- Assembly-like computational model
- Gap between theoretical algorithms and practical needs

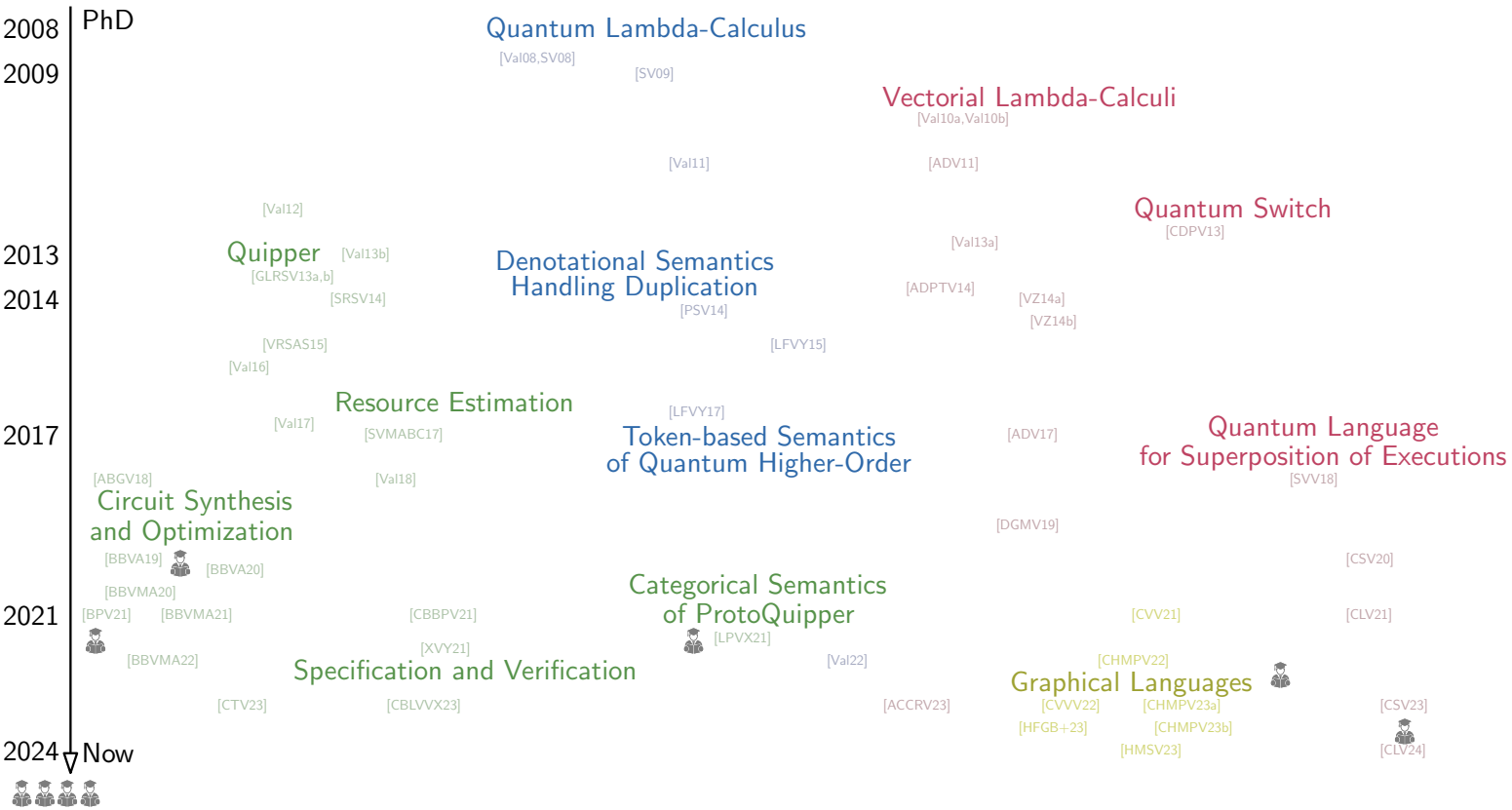
To

- Expressive high-level languages
- With sound semantics
- Usable for practical applications
- Amenable to formal analysis and reasoning

My Contributions

An incremental series of languages

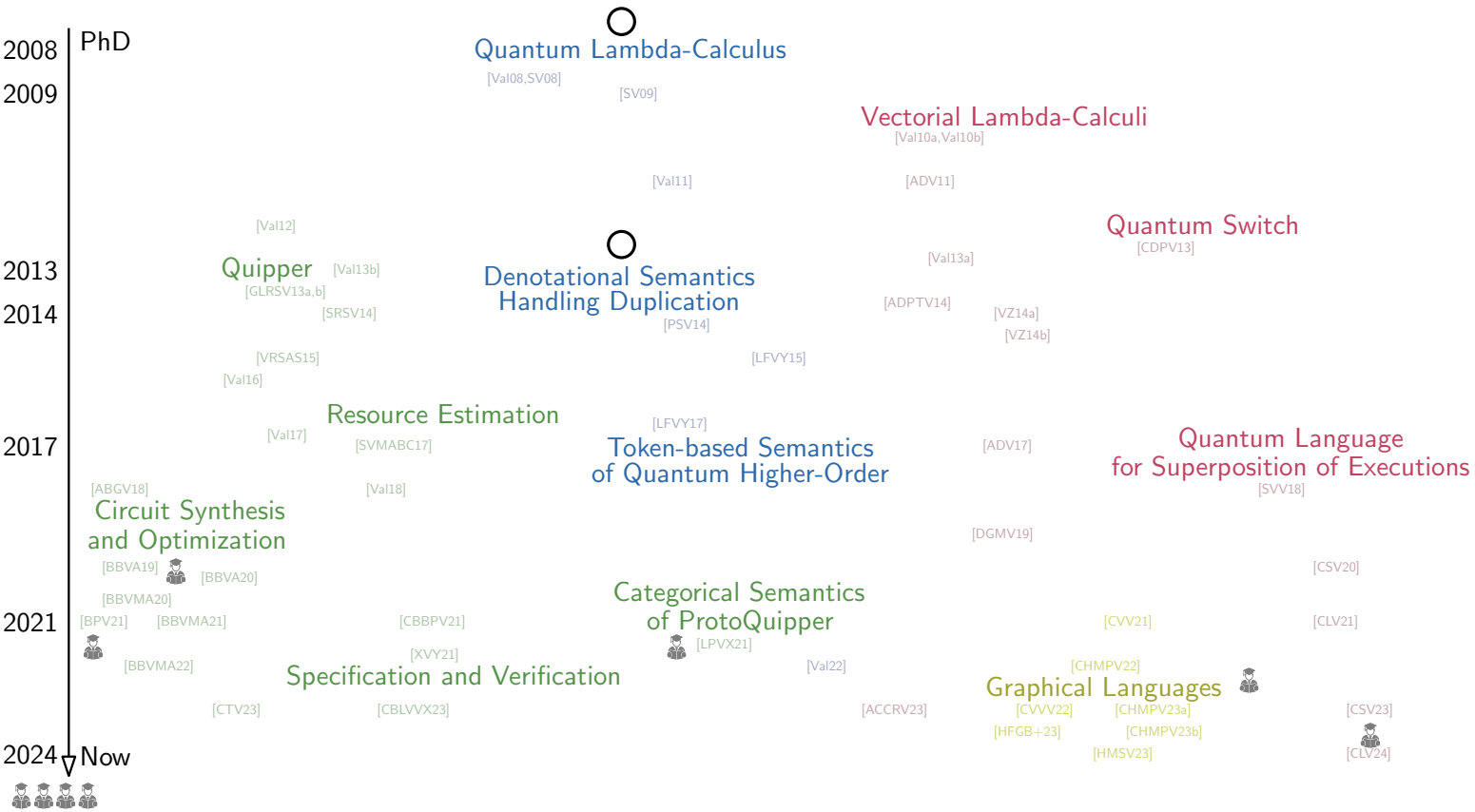
- » Towards increasingly fine-tuned expressivity
- » Soundness properties, analysis and reasoning tools



My Contributions

An incremental series of languages

- » Towards increasingly fine-tuned expressivity
- » Soundness properties, analysis and reasoning tools



Foundational Language for Quantum Computation

Based on Lambda-Calculus

- Core of all functional programming language:
  "Functions as first-class citizens"

$$(\lambda x.\texttt{body-of-function})(\texttt{argument})$$

- Amenable to side-effects: external memory, probability, etc.
- Type-systems to express program properties

Semantics

- Operational: evolution through the computation
- Denotational: mathematical representation

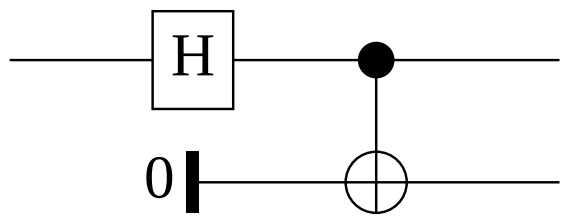
Soundness to certify behavior and structure

Quantum Circuits as Functions

Wires

- One wire: one qbit, encapulated in a type `qbit`
- Several wires: array of qbits, one per wire
  $\texttt{qbit} \otimes \texttt{qbit} \otimes \cdots \otimes \texttt{qbit}$

Circuits



- Inputs one qbit
- Outputs a pair of qbits
- Function from `qbit` to $\texttt{qbit} \otimes \texttt{qbit}$
- Side-effect: gates acting on the quantum memory

Quantum Lambda-Calculus

Terms

- Pairing constructs and fixpoints
- Boolean `true` and `false`, if-then-else
- Constant, opaque terms: `qinit`, `measure`, `H`, `CNOT`, ...
- Quantum states not in the language

Operational semantics

- Abstract machine encapsulating the quantum memory:

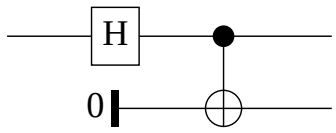
$$\Big(\underbrace{\tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle)}_{\text{state vector}}, \quad \underbrace{|xy\rangle}_{\text{“linking function”}}, \quad \underbrace{\lambda f.f\langle x, y\rangle}_{\text{lambda-term}}\Big)$$

- Call-by-value reduction strategy
- Quantum operations through the reduction strategy

Quantum Lambda-Calculus

Some terms are duplicable

- Boolean constants: `true`
- Regular, pure lambda-terms: $\lambda x.x$
- Circuit-descriptions: $\lambda x.(\text{let } z = \texttt{H}\, x \text{ in } \texttt{CNOT}\, \langle z, \texttt{qinit false}\rangle)$



Some terms are not duplicable

- Qbits: `H (qinit false)`
- Tuples containing a qbit: $\langle \lambda x.x,$ `H (qinit false)`$\rangle$
- Functions containing a qbit: let $y =$ `H (qinit false)` in $\lambda f.f\, y$

Distinction between

- Procedure for generating a qbit: duplicable
- End result of the procedure: qbit value, non-duplicable

Quantum Lambda-Calculus

Type system

$$A, B \ ::= \ \texttt{qbit} \mid \texttt{bit} \mid \top \mid A \otimes B \mid A \multimap B \mid !(A \multimap B)$$

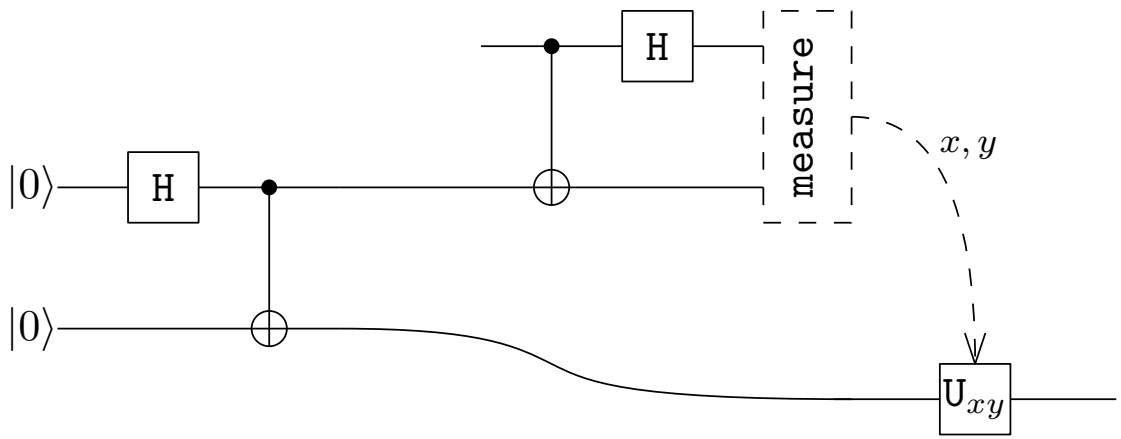
- Based on linear logic
- Non-duplicable functions with $A \multimap B$
- Duplicable functions with $!(A \multimap B)$
- Quantum operations are e.g. $\texttt{measure} : !(\texttt{qbit} \multimap \texttt{bit})$

Non-trivial mix

- Classical and quantum data
- General recursion in a probabilistic setting
- Entanglement at higher-order

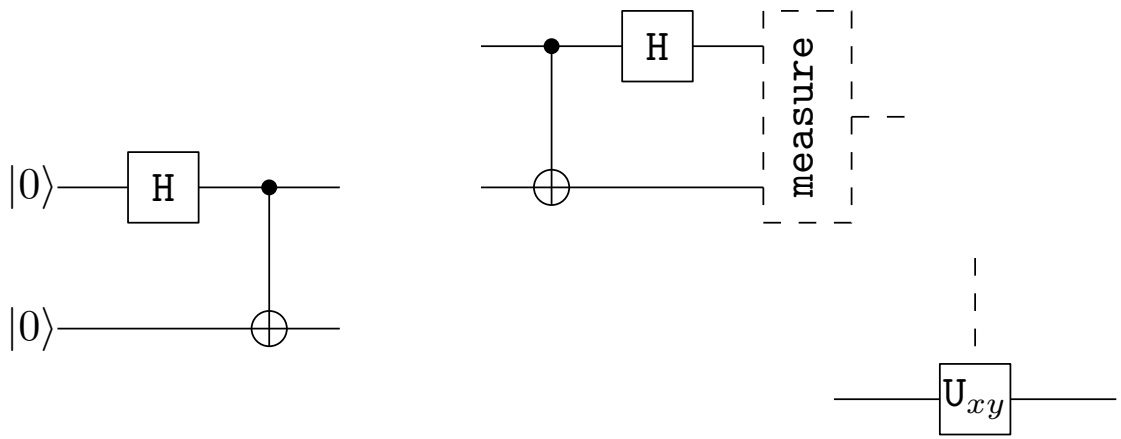
Quantum Lambda-Calculus

Example of higher-order entanglement: Teleportation



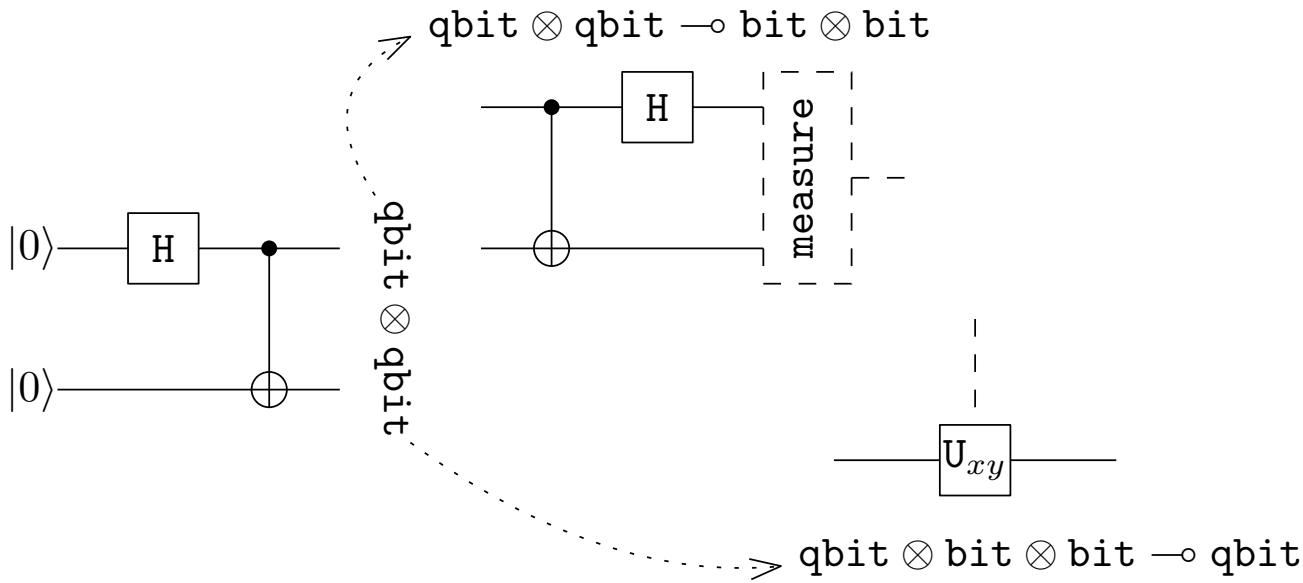
Quantum Lambda-Calculus

Example of higher-order entanglement: Teleportation



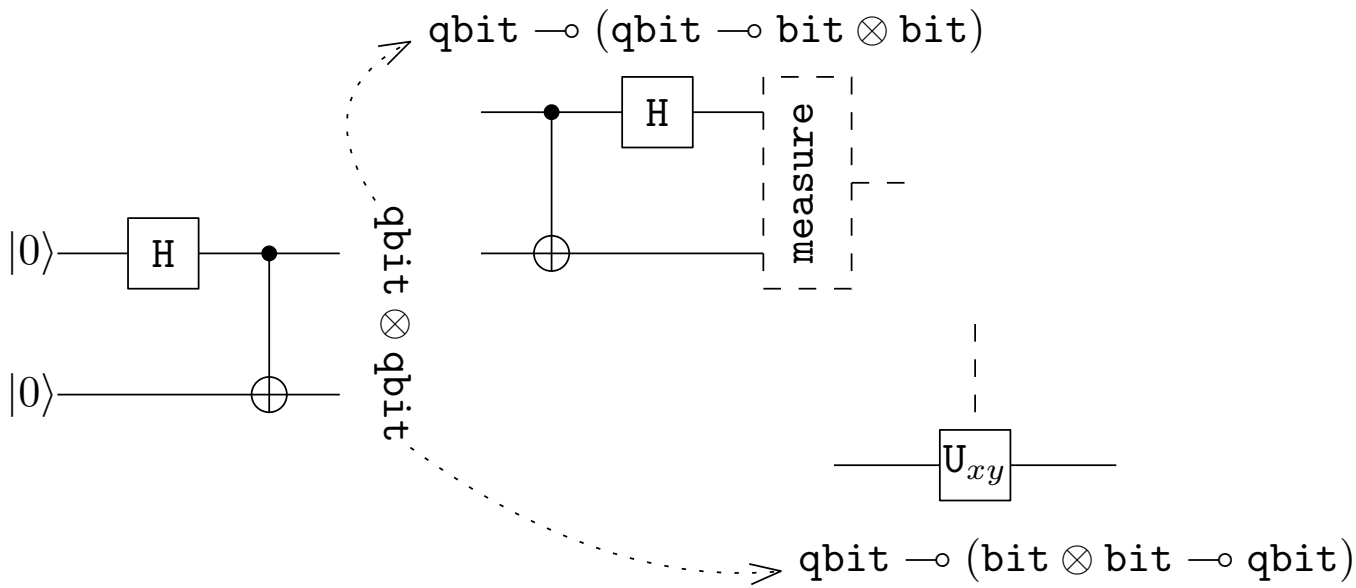
Quantum Lambda-Calculus

Example of higher-order entanglement: Teleportation



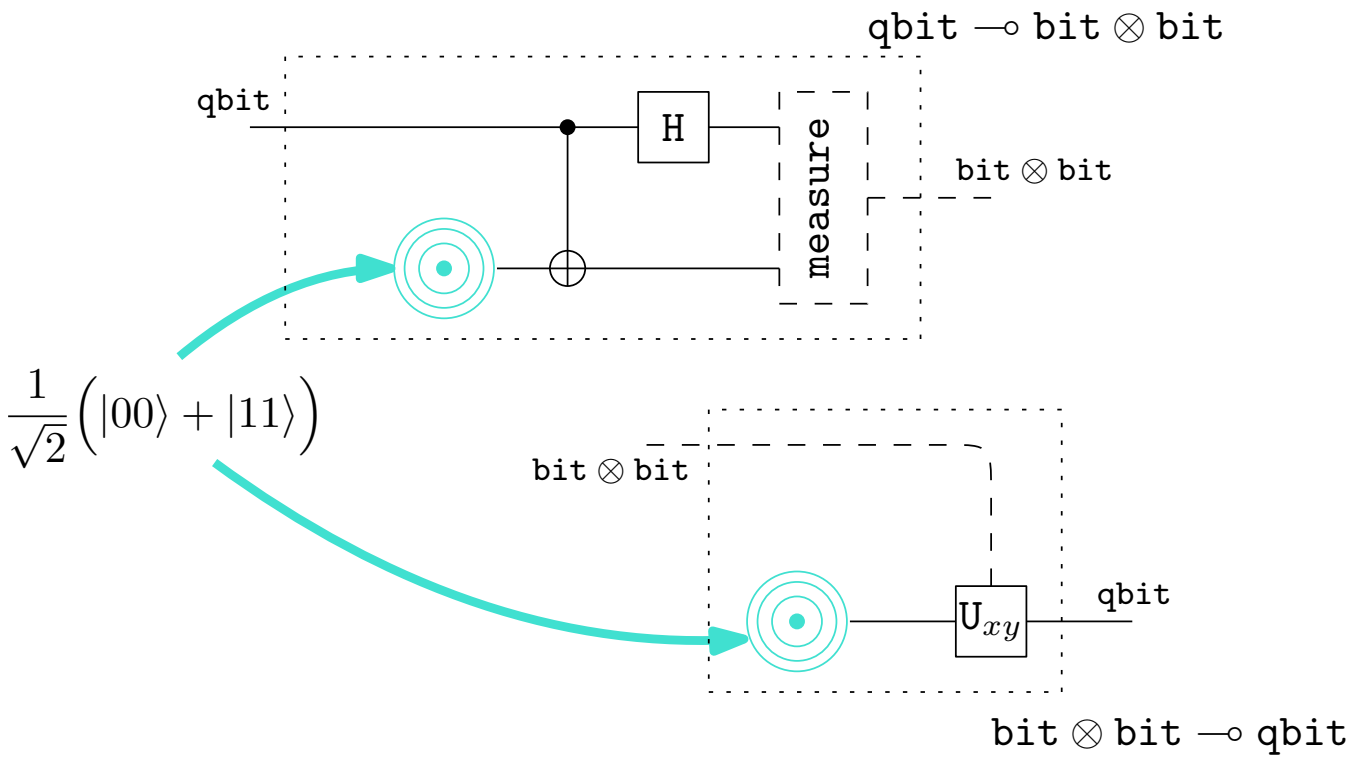
Quantum Lambda-Calculus

Example of higher-order entanglement: Teleportation



Quantum Lambda-Calculus

Example of higher-order entanglement: Teleportation



A pair of two entangled non-duplicable functions

$$(\texttt{qbit} \multimap \texttt{bit} \otimes \texttt{bit}) \otimes (\texttt{bit} \otimes \texttt{bit} \multimap \texttt{qbit})$$

Quantum Lambda-Calculus

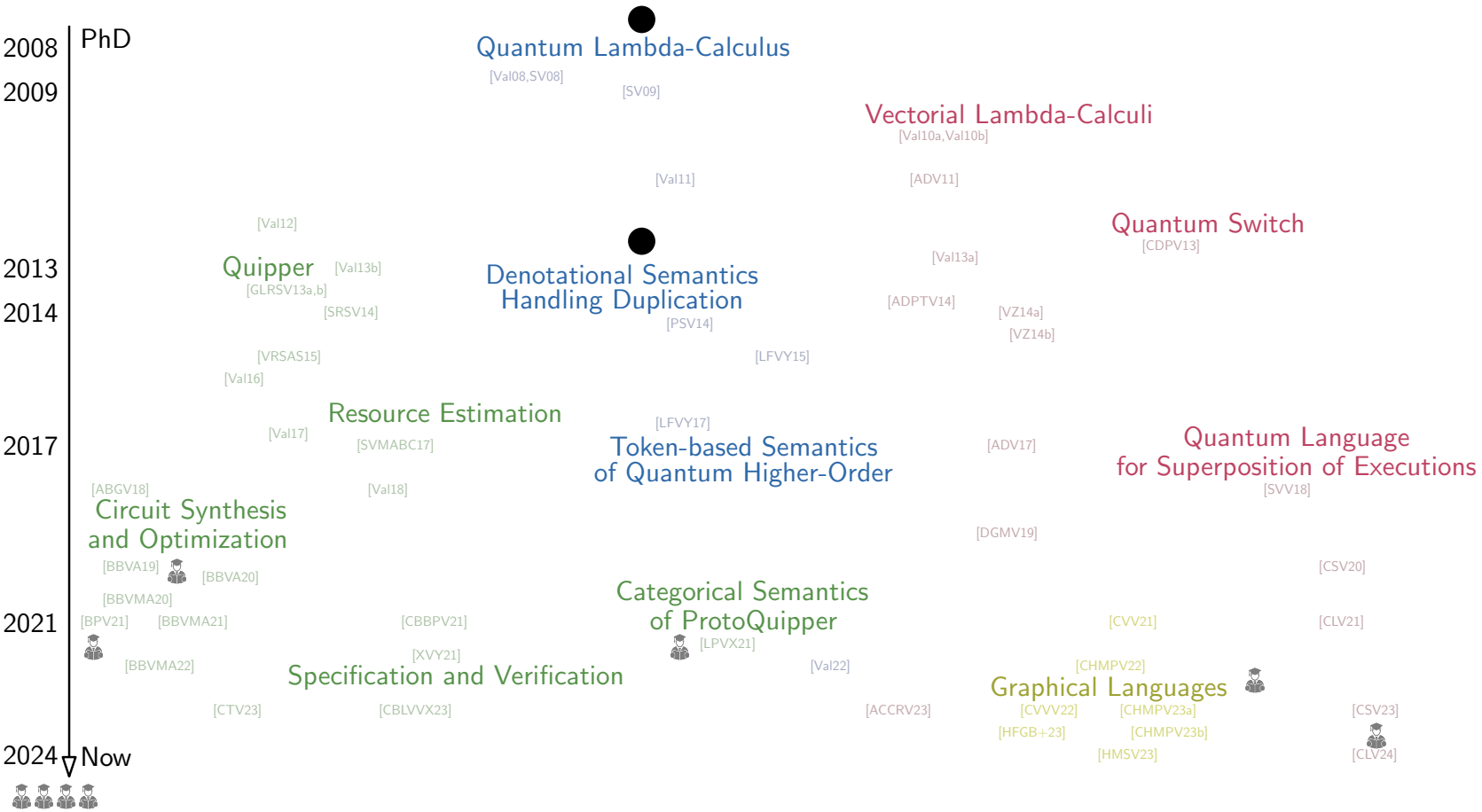
Denotational Semantics

- Ensuring soundness of arbitrary higher-order entanglement
- When mixed with duplication and non-termination
- Using a mathematical characterization of programs

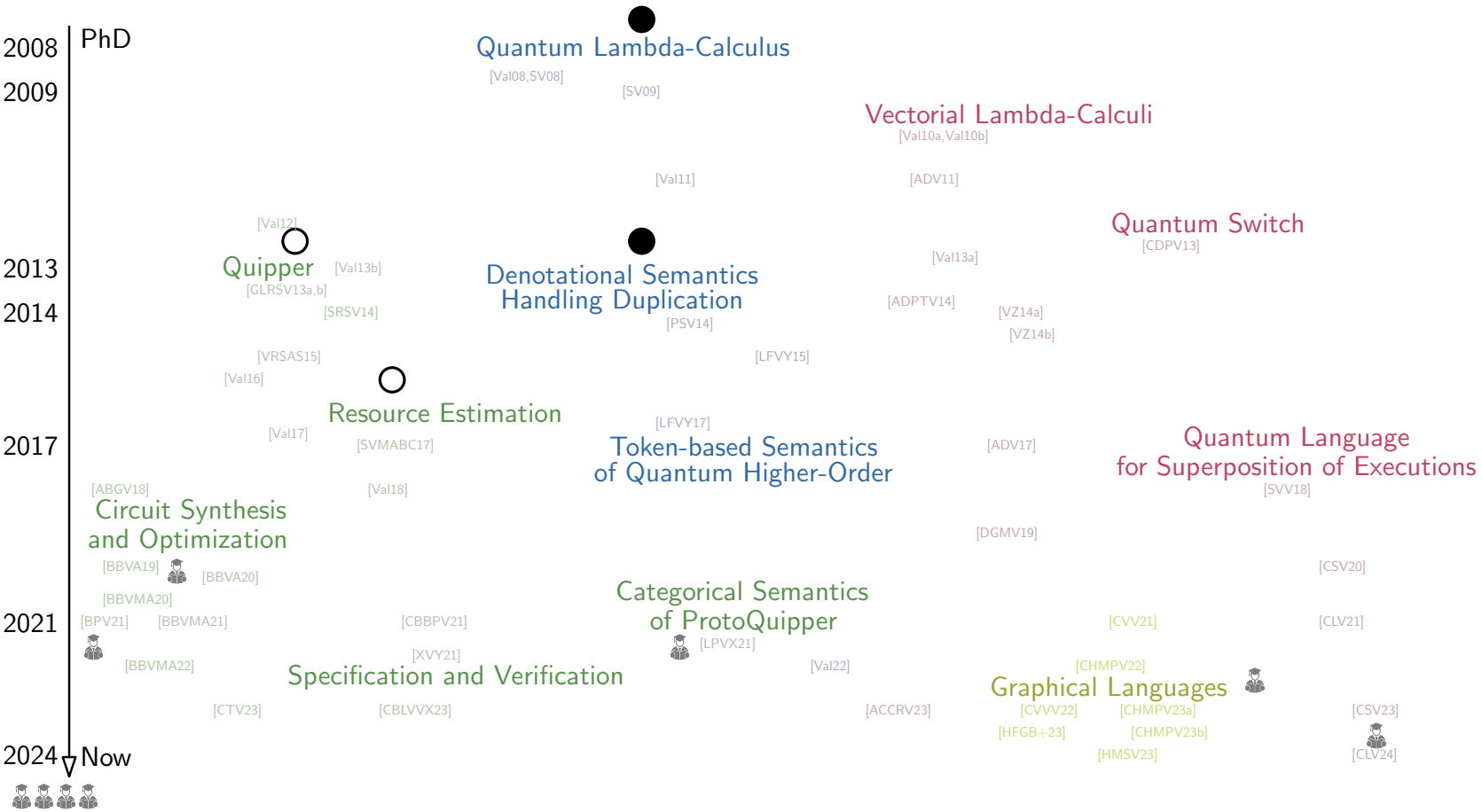
History of the construction

- Semantics for the linear fragment based on CPM [SV08]
  - Linear maps in finite dimension
  - Handling duplicability requires infinite dimension
- In 2013: Semantics for a higher-order probabilistic language
- Building on the same idea
  - Semantics for all of the quantum lambda-calculus [PSV14]
  - Categorical construction on top of CPM
  - Handles both duplicable and non-duplicable data
  - Proved the semantics matches the execution of the program

Quantum Lambda-Calculus



Quantum Lambda-Calculus



Quipper

Before Quipper

- No quantum programming languages
- Algorithms with theoretical potential

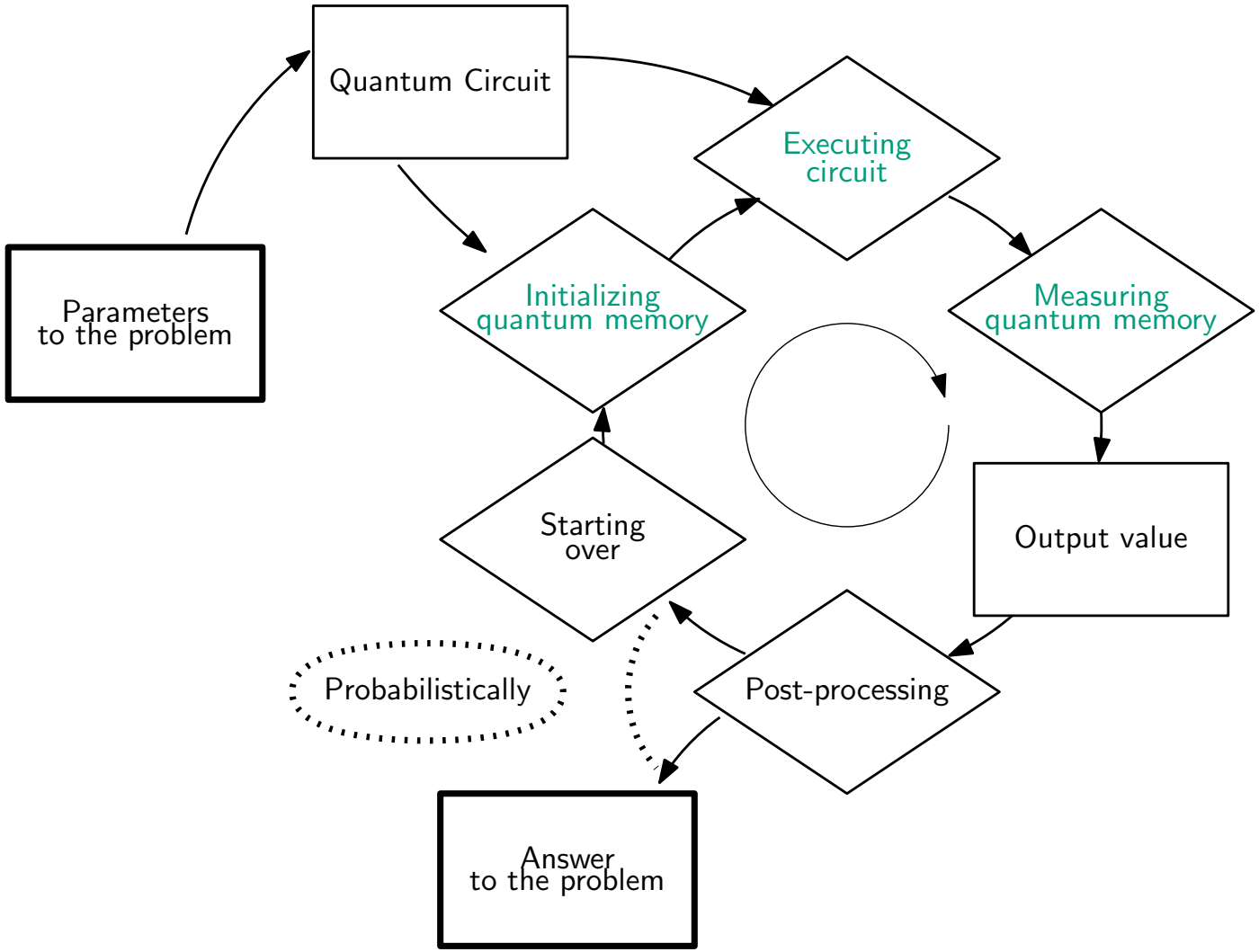
Contributions [GLRS**V**13b]

- Domain specific language embedded in Haskell
- Aimed at programming quantum algorithms for real
- Fitted for the analysis of quantum programs

Design principle

- Extension of the quantum lambda-calculus
- Circuit-description language
- Relies on Haskell's monad construction for handling circuits

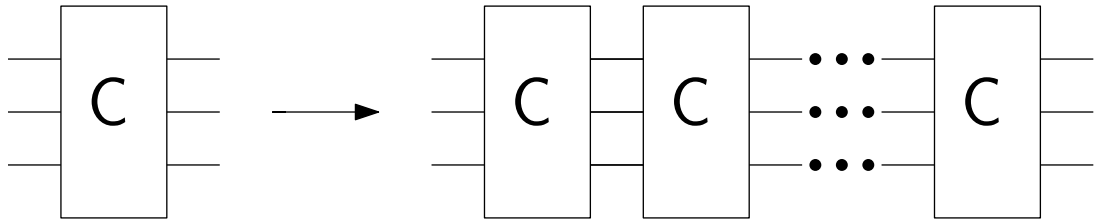
Structure of Quantum Algorithms



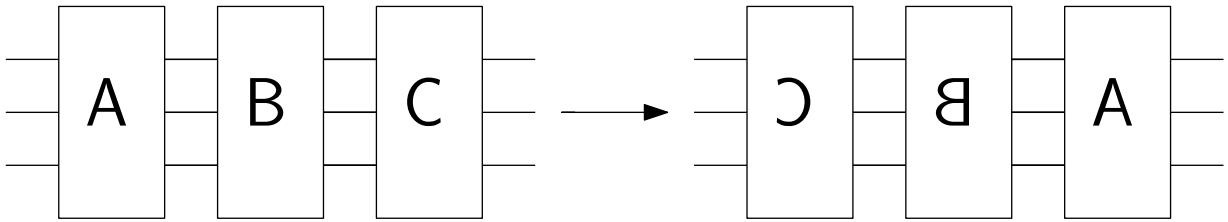
Structure of Quantum Algorithms

Circuit combinators: used to build complex circuits

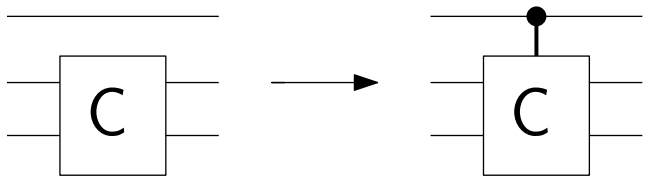
- Sequence: simple, low-level chaining of elementary gates
- Repetition



- Inversion

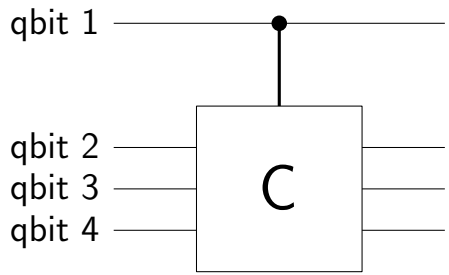


- Control



Structure of Quantum Algorithms

Controlled circuit : conditional action of a circuit



C is applied on qbits 2-4 only when qbit 1 is true:
Suppose that C flips its input bits. Then the previous circuit does

$$\begin{array}{rl} \text{qbit} & 1\,234 \\ \frac{1}{\sqrt{2}} & |1\,010\rangle \\ + \frac{1}{\sqrt{2}} & |0\,110\rangle \end{array} \quad \longmapsto \quad \begin{array}{rl} \text{qbit} & 1\,234 \\ \frac{1}{\sqrt{2}} & |1\,101\rangle \\ + \frac{1}{\sqrt{2}} & |0\,110\rangle \end{array}$$

This acts as a "quantum test"

» Unlike a if-then-else, control qbit still exists after the test

Quipper: a Circuit Description Language

Circuits as functions

- Quantum lambda-calculus: `H` : `qbit` $\multimap$ `qbit`
  - Side-effect: apply `H` on the quantum memory
  - Encapsulated by the operational semantics
- Quipper: `H : Qbit -> Circ Qbit`
  - Inputs a value of type `Qbit`
  - Outputs a computation of type `Qbit`
  - The computation buffers a circuit snippet
- The type constructor `Circ` is a monad
  - Standard technique in Haskell
  - Circuit snippet programmatically accessible

Quipper: a Circuit Description Language

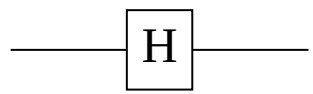
Procedural presentation of circuits

```
prog :: Qbit -> Circ (Qbit,Qbit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```

Quipper: a Circuit Description Language

Procedural presentation of circuits

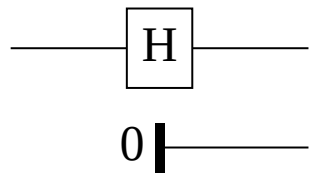
```
prog :: Qbit -> Circ (Qbit,Qbit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Quipper: a Circuit Description Language

Procedural presentation of circuits

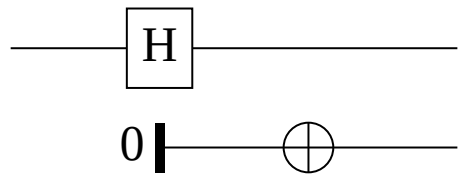
```
prog :: Qbit -> Circ (Qbit,Qbit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Quipper: a Circuit Description Language

Procedural presentation of circuits

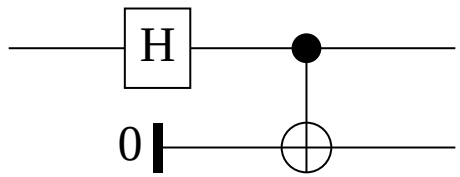
```
prog :: Qbit -> Circ (Qbit,Qbit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Quipper: a Circuit Description Language

Procedural presentation of circuits

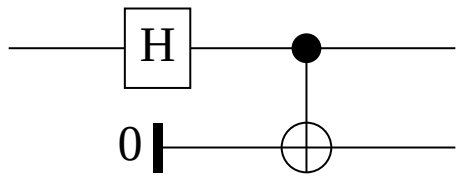
```
prog :: Qbit -> Circ (Qbit,Qbit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



Quipper: a Circuit Description Language

Procedural presentation of circuits

```
prog :: Qbit -> Circ (Qbit,Qbit)
prog q = do
  hadamard_at q
  r <- qinit False
  qnot_at r `controlled` q
  return (q,r)
```



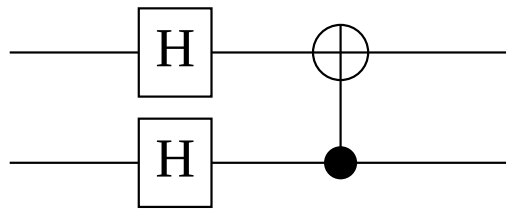
Quipper: a Circuit Description Language

High-level circuit combinators

- Iteration, circuit inversion, control, etc.

Example

```
prog :: (Qbit,Qbit) -> Circ (Qbit,Qbit)
prog (p,q) = do
  hadamard_at p
  hadamard_at q
  qnot_at p `controlled` q
  return (p,q)
```



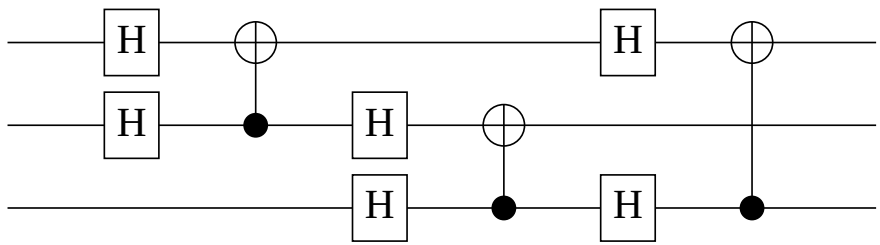
Quipper: a Circuit Description Language

High-level circuit combinators

- Iteration, circuit inversion, control, etc.

Example

```
prog :: (Qbit,Qbit) -> Circ (Qbit,Qbit)
...

prog2 :: (Qbit,Qbit,Qbit) -> Circ ()
prog2 (p,q,r) = do
  prog (p,q)
  prog (q,r)
  prog (p,r)
  return ()
```



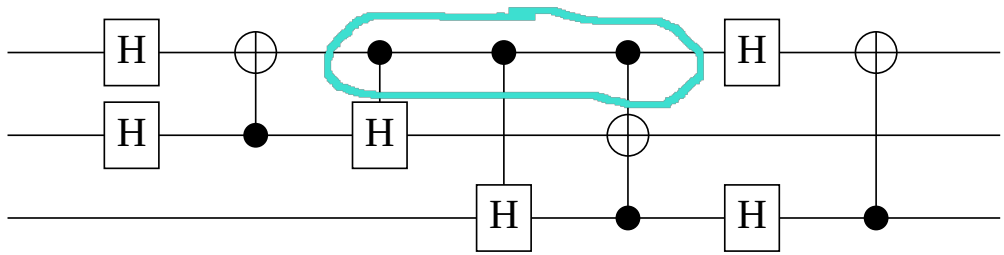
Quipper: a Circuit Description Language

High-level circuit combinators

- Iteration, circuit inversion, control, etc.

Example

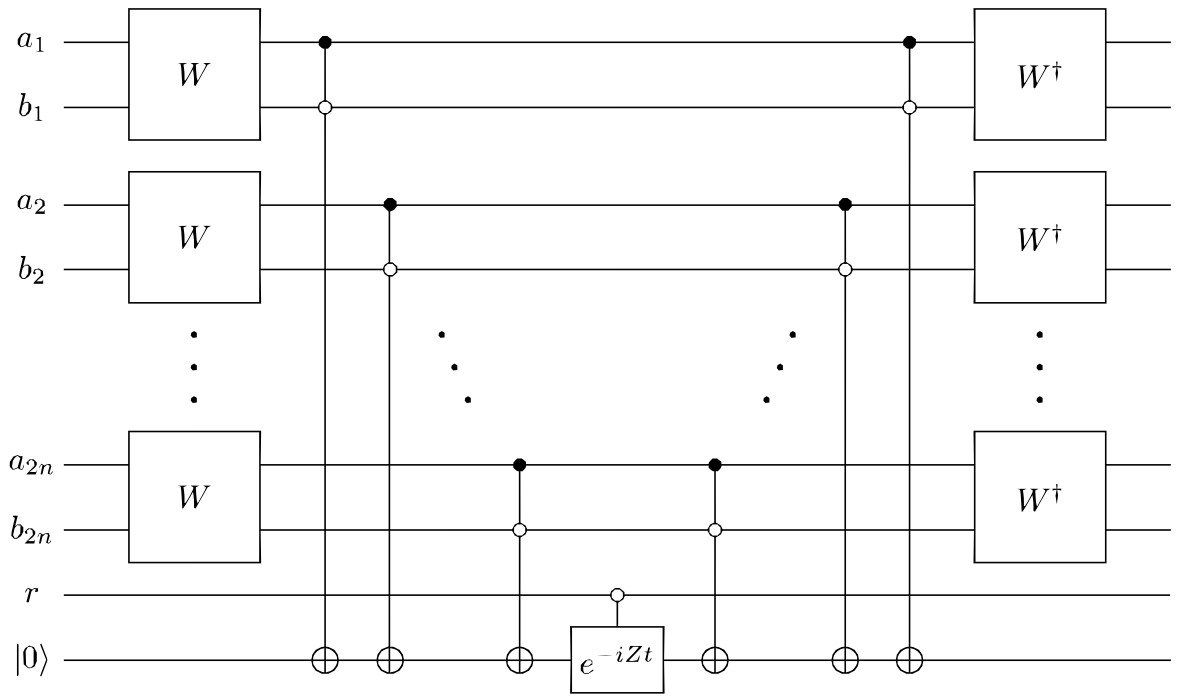
```
prog :: (Qbit,Qbit) -> Circ (Qbit,Qbit)
...

prog2 :: (Qbit,Qbit,Qbit) -> Circ ()
prog2 (p,q,r) = do
  prog (p,q)
  prog (q,r) `controlled` p
  prog (p,r)
  return ()
```



One of Quipper's Features: Parametric Circuits

Circuits are built from the parameters of the problem

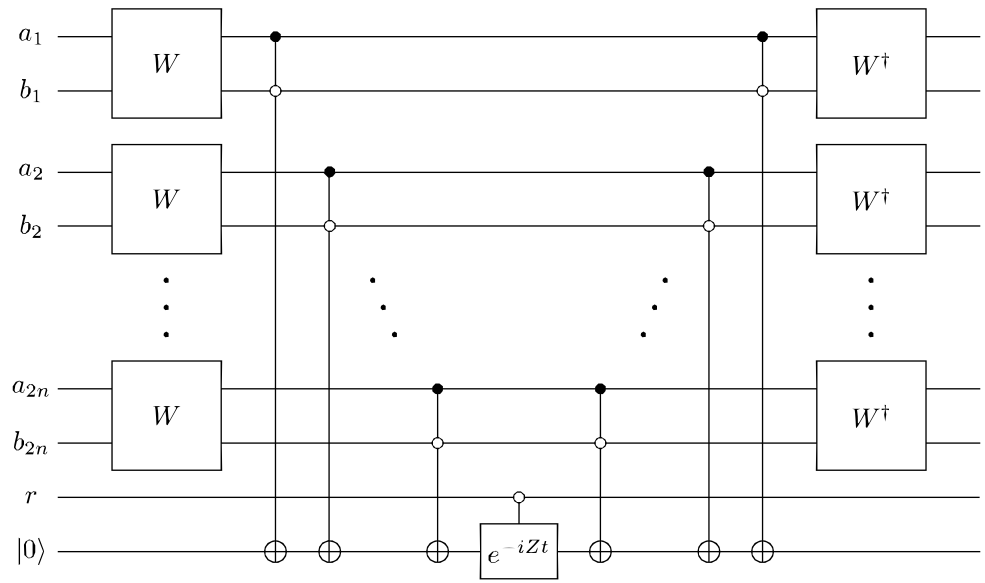
- Their structure depends on these parameters



One of Quipper's Features: Parametric Circuits

Types for Families of Circuits.

- Type constructor for lists: `[Qbit]`
- Circuit



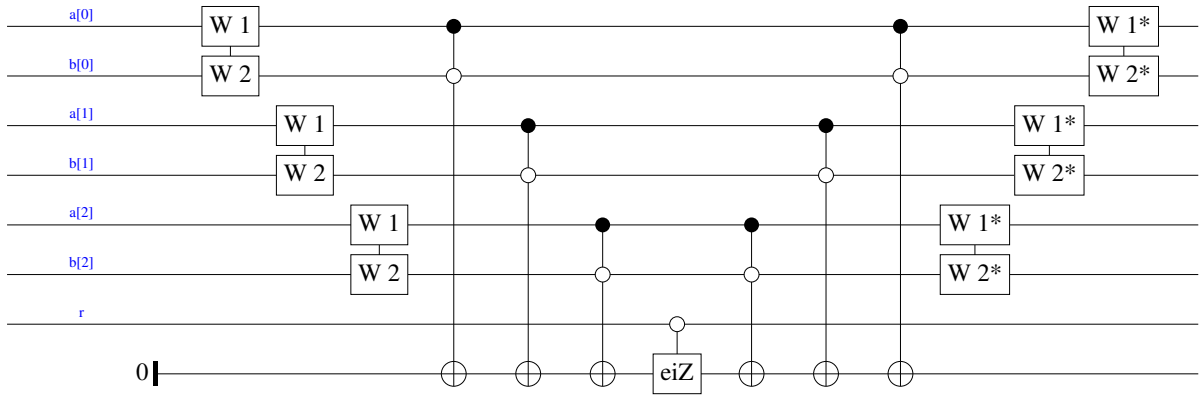
typed with

```
([Qbit],[Qbit],Qbit) -> ([Qbit],[Qbit],Qbit,Qbit)
```

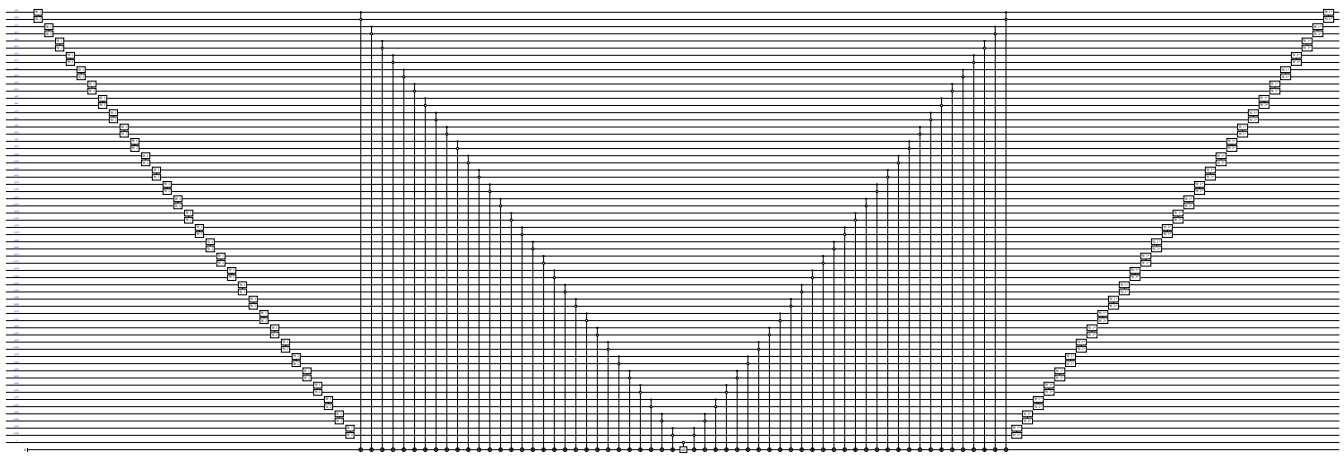
One of Quipper's Features: Parametric Circuits

Produced with Quipper

» $n = 3$



» $n = 30$



Quipper a Practical Analysis Tool

Example: Resource Estimation using Quipper

» Concrete implementation of seven algorithms [GLRS**V**13b]

Focus on QLSA: Quantum linear system algorithm

» Linear algebra problem: Given $\vec{v}$, $\vec{b}$ and A
  – Compute $\langle \vec{x} \mid \vec{v} \rangle$ with x such that $A \cdot \vec{x} = \vec{b}$

» Implementation following (Clader *et al.*, 2013) [S**V**MABC17]

» Complexity depending on several parameters

κ	condition number	d	sparseness of A
N	size of the matrix A	ε	maximal error

» For $\kappa = 10^4$, $d = 7$, $N \sim 10^{10}$, $\varepsilon = 10^{-2}$
  – $\tilde{O}(\kappa d^2 \log(N)/\varepsilon^2)$ gives: $\sim 10^{12}$ gates
  – Actual count: 10^{29} gates

Takeaway Complexity $\neq$ Concrete counts

» Importance of program analysis

Quipper: Summary of my contributions

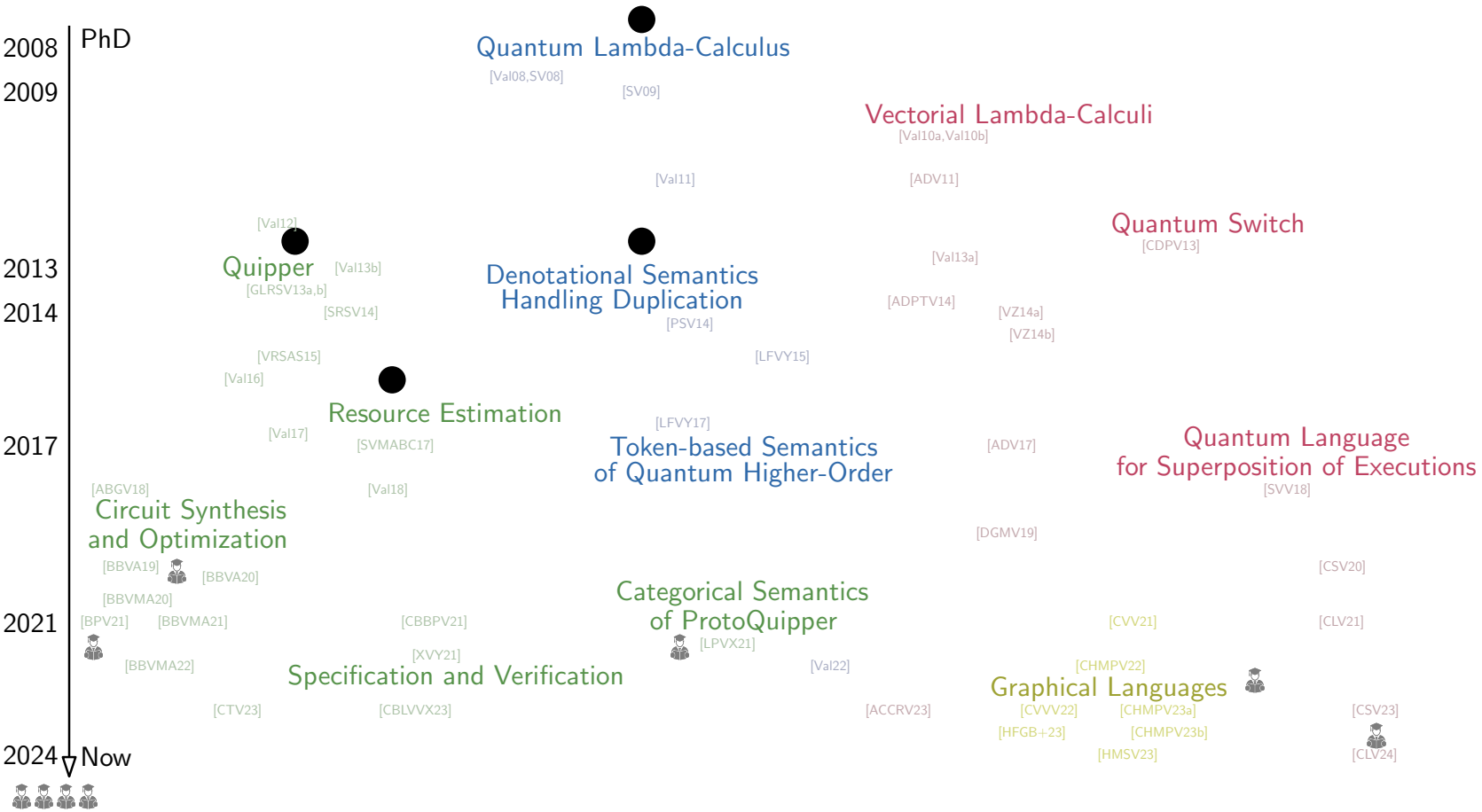
Foundational work

- First scalable quantum programming language [GLRS**V**13b]
- Language design principles for circuit-description [**V**RSAS15]
- Resource estimation for concrete use-cases [S**V**MABC17]
- Categorical analysis of the fragment ProtoQuipper [LP**V**X21, 🎓]

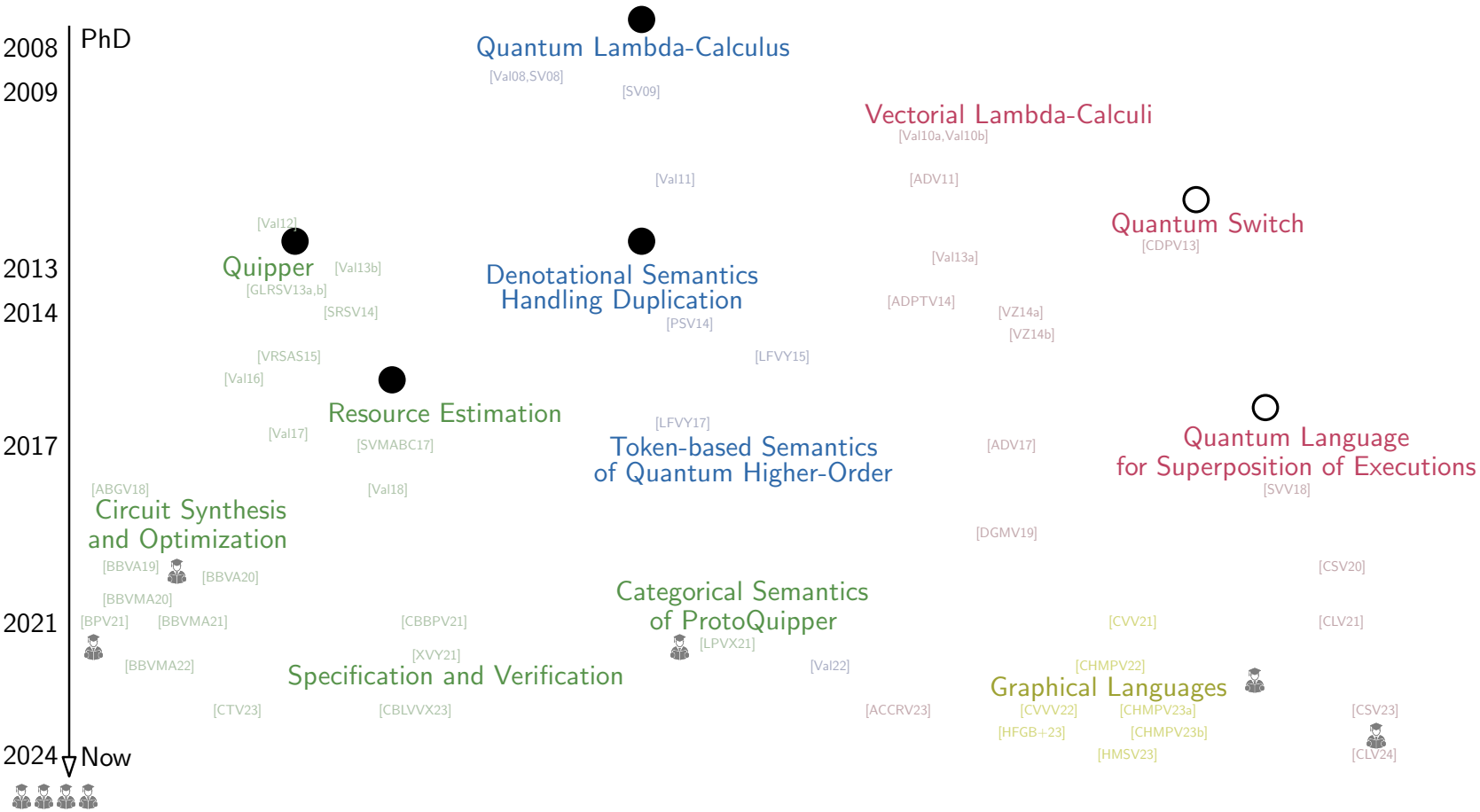
Subsequent, related works

- Oracle synthesis from classical description [Val16]
- Compilation techniques using ZX calculus [BP**V**21, 🎓]
- Circuit synthesis & optimization [ABG**V**18, BB**V**A19,BB**V**MA20, 🎓]
- Quantum specification and verification [CBBP**V**21,CBL**V**VX21]

Quipper: Summary of my contributions



Quipper: Summary of my contributions



Limit of Circuits: Superposition of Executions

Quantum Switch [CDP**V**13]

- Consider A and B two unitary maps
- Define the higher-order linear map `Switch` by

$$\texttt{Switch}\, A\, B\, (|1\rangle \otimes |q\rangle) = |1\rangle \otimes (A \circ B\, |q\rangle)$$
$$\texttt{Switch}\, A\, B\, (|0\rangle \otimes |q\rangle) = |0\rangle \otimes (B \circ A\, |q\rangle)$$

- The function is linear

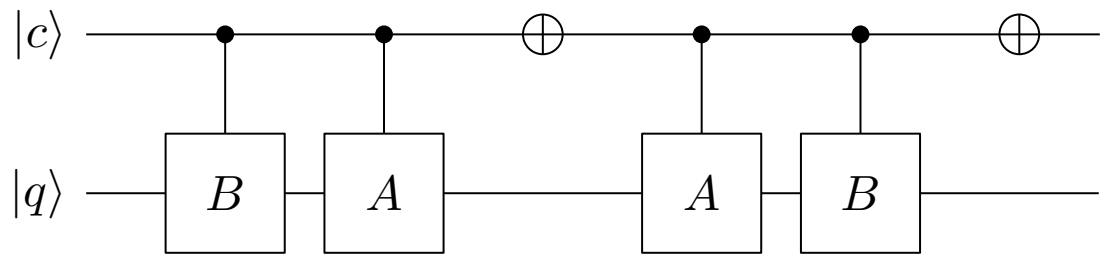
$$\texttt{Switch}\, A\, B \left(\alpha \cdot |0\rangle \otimes |q_0\rangle + \beta \cdot |1\rangle \otimes |q_1\rangle\right) =$$
$$\alpha \cdot |0\rangle \otimes (B \circ A\, |q_0\rangle) + \beta \cdot |1\rangle \otimes (A \circ B\, |q_1\rangle)$$

- Superposition of executions of $A \circ B$ and $B \circ A$.

Limit of Circuits: Superposition of Executions

Realizing the quantum Switch

» If two copies of A and B: using circuit combinators

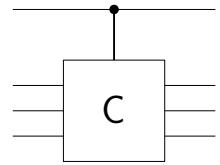


» If only a single copy of each
- Not expressible in the circuit model [CDP**V**13]
- Physically realizable (with quantum linear optics)

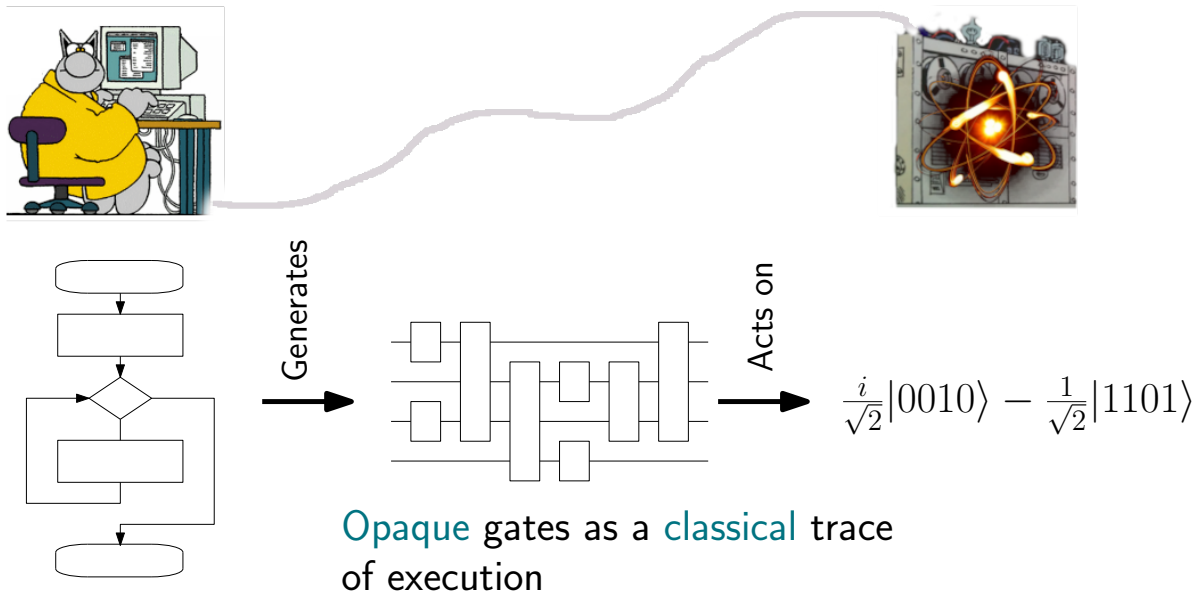
Notions of Control Flow

Control flow in quantum computation has two meanings

» The (quantum) control of a gate



» The (classical) control-flow of the program



Opaque gates as a classical trace of execution

Notions of Control Flow

Goal Extend gate control into a full fledged control flow

- From quantum Switch to general quantum tests
- Purely quantum loops, iteration or recursion
- Purely quantum function calls
- Programming primitives to describe gates and combinators

My contributions A purely quantum language [SVV18]

- Extension of (Proto)Quipper:
  - Syntax to describe quantum superposition
  - Lambda-calculus to manipulate circuits
- Higher-order syntax to program
  - Low-level quantum gates
  - High-level circuit combinators
- With soundness properties

Purely Quantum Control Flow

Example of purely quantum control

- Hadamard operation

$$\mathtt{H}: \quad \begin{array}{l} |0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{array}$$

- A "map" operation with mock-up type

$$\mathtt{map} : (\mathtt{qbit} \multimap \mathtt{qbit}) \to (\mathtt{list}(\mathtt{qbit}) \multimap \mathtt{list}(\mathtt{qbit}))$$

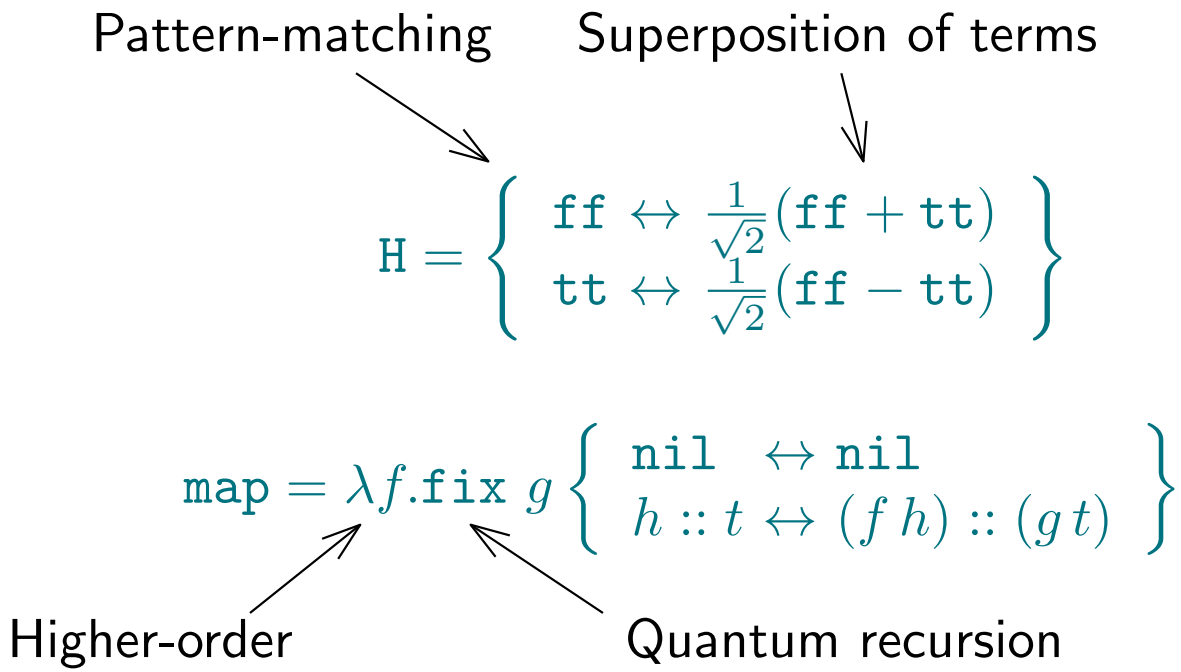
- The operation $\mathtt{map}\,\mathtt{H}$ can then be applied on

$$\frac{1}{\sqrt{2}} \cdot \left| [0,0] \right\rangle \quad + \quad \frac{1}{\sqrt{2}} \cdot \left| [0,0,1,0] \right\rangle$$

The number of iterations differs for each list

Purely Quantum Control Flow

Quantum Pattern Matching A syntax for quantum control



With soundness results [SVV18]

Purely Quantum Control Flow

Summary: Developments I took part in

- Introduction of the quantun Switch [CDP**V**13]
- Purely quantum languages
  - Vectorial lambda-calculi [AD**V**17,DGM**V**19]
  - Quantum pattern-matching [S**V**V18]
- Curry-Howard correspondence using μMALL [CS**V**23, 🎓]
- Development of categorical semantics [CL**V**21, 🎓]

Conclusion

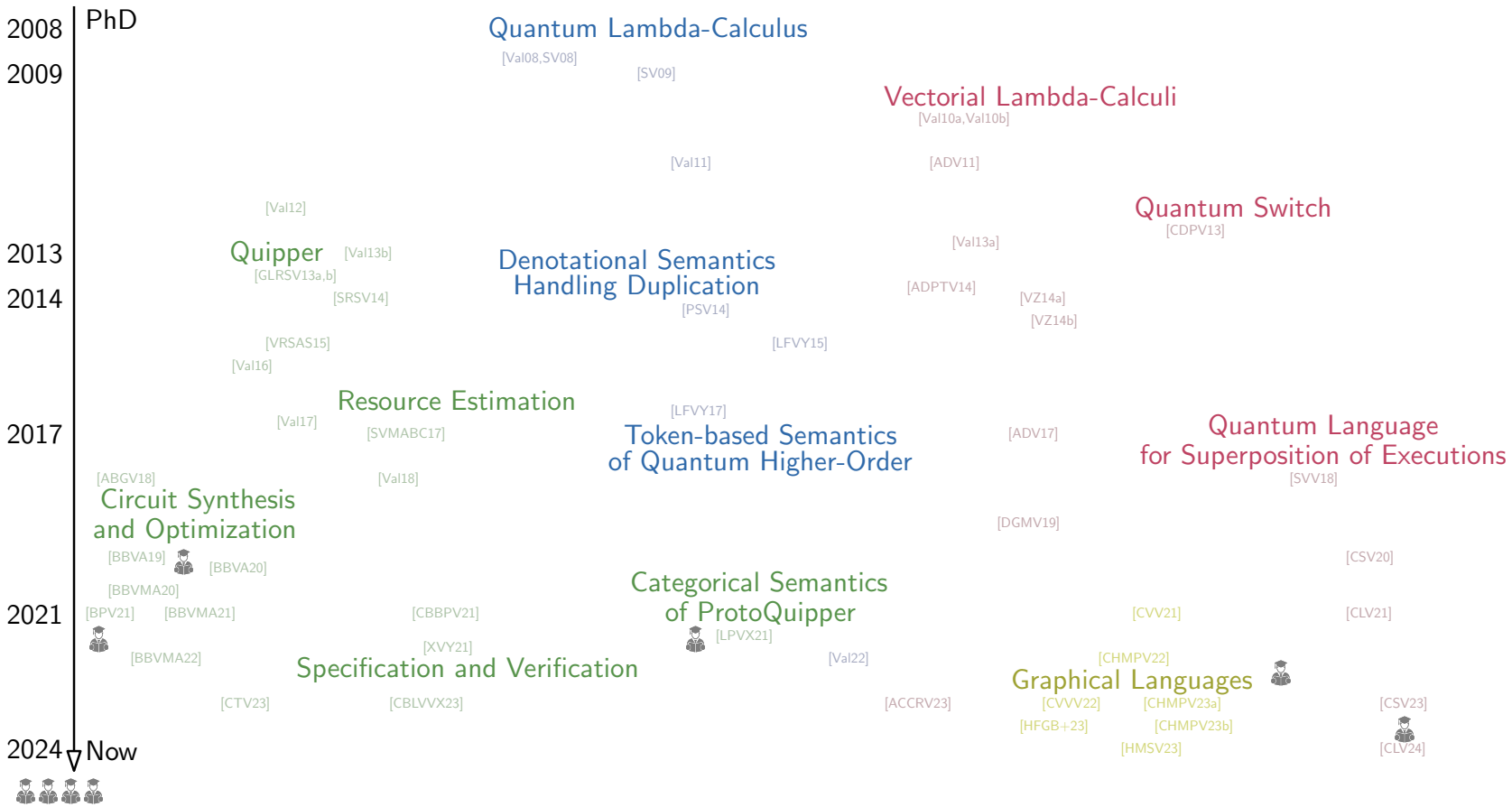
Summary: Towards expressivity in quantum languages

- Quantum Lambda-Calculus
  - No circuit manipulation
- Quipper
  - Adds: circuit manipulation
  - But gates and combinators opaque
- Quantum Pattern Matching
  - Adds: Syntax to program gates and circuit combinators

Current research

- Unification of quantum and classical control
- Graphical languages as circuit abstraction
- Quantum compilation toolchain
- Static analysis of quantum programs

Conclusion


2008
PhD
2009
2013
2014
2017
2021
2024
Now
Quantum Lambda-Calculus
[Val08,SV08]
[SV09]
Vectorial Lambda-Calculi
[Val10a,Val10b]
[Val11]
[ADV11]
[Val12]
Quantum Switch
[CDPV13]
[Val13a]
Quipper
[Val13b]
[GLRSV13a,b]
[SRSV14]
Denotational Semantics
Handling Duplication
[PSV14]
[ADPTV14]
[VZ14a]
[VZ14b]
[VRSAS15]
[LFVY15]
[Val16]
Resource Estimation
[LFVY17]
[Val17]
[SVMABC17]
Token-based Semantics
of Quantum Higher-Order
[ADV17]
Quantum Language
for Superposition of Executions
[SVV18]
[ABGV18]
[Val18]
Circuit Synthesis
and Optimization
[DGMV19]
[BBVA19]
[BBVA20]
[CSV20]
[BBVMA20]
Categorical Semantics
of ProtoQuipper
[BPV21]
[BBVMA21]
[CBBPV21]
[CVV21]
[CLV21]
[LPVX21]
[XVY21]
[BBVMA22]
[Val22]
[CHMPV22]
Specification and Verification
Graphical Languages
[CTV23]
[CBLVVX23]
[ACCRV23]
[CVVV22]
[CHMPV23a]
[CSV23]
[HFGB+23]
[CHMPV23b]
[HMSV23]
[CLV24]