A HoTT Quantum Equational Theory

Jennifer Paykin
Galois, Inc
jpaykin@galois.com

MURI Project Review
University of Maryland

March 8, 2019

With Steve Zdancewic at the University of Pennsylvania.

Quantum data, classical control

...via embedded languages

Quantum data, classical control

...via embedded languages

- Quipper [Green et al., 2013]
  - Embedded in Haskell, a functional lazy language.
  - Uses Haskell types, functions, data structures, type classes, template haskell... to construct quantum circuits.
  - Access to Haskell REPL and debugging tools.

Quantum data, classical control

...via embedded languages

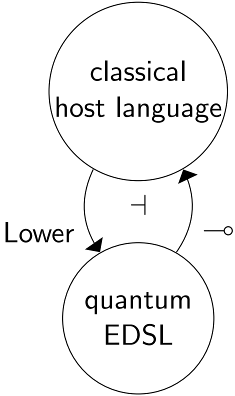
- Quipper [Green et al., 2013]
  - Embedded in Haskell, a functional lazy language.
  - Uses Haskell types, functions, data structures, type classes, template haskell... to construct quantum circuits.
  - Access to Haskell REPL and debugging tools.
- LiQUiD, Q language, Project Q, QISKit, pyQuill...

Quantum data, classical control

...via embedded languages

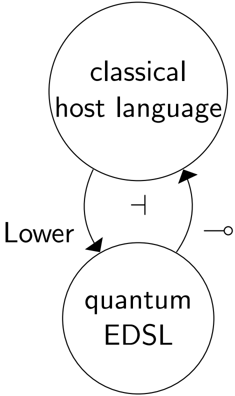
- Quipper [Green et al., 2013]
  - Embedded in Haskell, a functional lazy language.
  - Uses Haskell types, functions, data structures, type classes, template haskell... to construct quantum circuits.
  - Access to Haskell REPL and debugging tools.
- LiQUiD, Q language, Project Q, QISKit, pyQuill...
- $\mathcal{Q}$WIRE [Paykin et al., 2017, Rand et al., 2017]
  - A formal theory of embedded quantum circuits.
  - Implemented as an embedded language in Coq, a theorem prover with dependent types.
  - Uses Coq theorem proving capabilities to prove correctness of quantum circuits.

Quantum/non-quantum calculus



- Based on Linear/Non-Linear (LNL) logic [Benton, 1995]
- Linear types, pairs ($\otimes$), etc

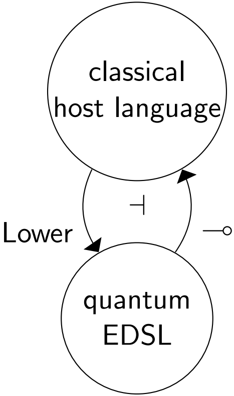
Quantum/non-quantum calculus



- Based on Linear/Non-Linear (LNL) logic [Benton, 1995]
- Linear types, pairs ($\otimes$), etc

$$\frac{a : \alpha}{\mathsf{put}\ a : \mathsf{QExp}\ \cdot\ (\mathsf{Lower}\ \alpha)}$$

Quantum/non-quantum calculus

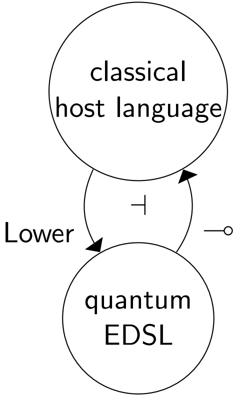


- Based on Linear/Non-Linear (LNL) logic [Benton, 1995]
- Linear types, pairs ($\otimes$), etc

$$\frac{a : \alpha}{\text{put } a : \text{QExp} \ \cdot \ (\text{Lower } \alpha)}$$

$$\frac{e : \text{QExp } \Delta \ (\text{Lower } \alpha) \qquad f : \alpha \to \text{QExp } \Delta' \ \tau}{e >!\ f : \text{QExp } (\Delta, \Delta') \ \tau}$$

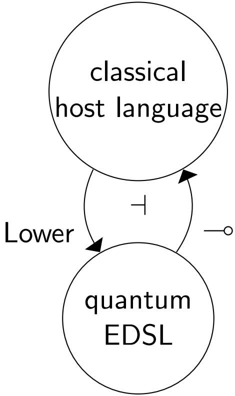
Quantum/non-quantum calculus



- Derived quantum operations:

$$\begin{aligned} \mathsf{Qubit} &= \mathsf{Lower}(\mathsf{Bool}) \\ |b\rangle &= \mathsf{put}\ b \\ \mathsf{let}\ b := \mathsf{meas}\ e\ \mathsf{in}\ e' &= e >!\ \lambda b.e' \end{aligned}$$

Quantum/non-quantum calculus



- Derived quantum operations:

$$\mathsf{Qubit} = \mathsf{Lower}(\mathsf{Bool})$$
$$|b\rangle = \mathsf{put}\ b$$
$$\mathsf{let}\ b := \mathsf{meas}\ e\ \mathsf{in}\ e' = e >!\ \lambda b.e'$$

- Unitaries (not derived):

$$\frac{U : \mathsf{UMatrix}(\sigma, \tau) \qquad e : \mathsf{QExp}\ \Delta\ \sigma}{U \# e : \mathsf{QExp}\ \Delta\ \tau}$$

Reasoning about quantum data

- Denotational semantics
  - Spaces are exponential in size of program

Reasoning about quantum data

- Denotational semantics
  - Spaces are exponential in size of program
- Program logics
  - Best suited to imperative quantum languages

Reasoning about quantum data

- Denotational semantics
  - Spaces are exponential in size of program
- Program logics
  - Best suited to imperative quantum languages
- Equational theory
  - Syntactic rules that characterize when programs are equivalent.
  - May or may not be directed; difficult to normalize.
  - Validated with respect to denotational semantics.

Goal

Equational theory for *embedded* quantum circuit language.

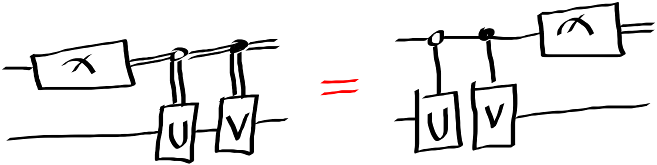
Goal

Equational theory for *embedded* quantum circuit language.

- Interaction between quantum data and host language control
- NOT equational theory for classes of unitaries

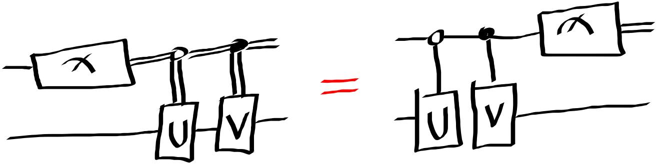
Prior work – Staton [2015]

- Equational theory for algebra with unitaries and classical control.



Prior work – Staton [2015]

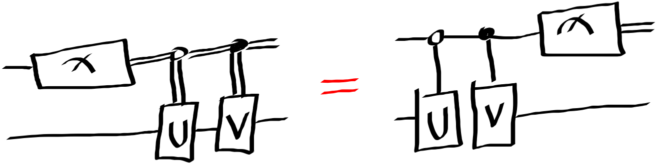
- Equational theory for algebra with unitaries and classical control.



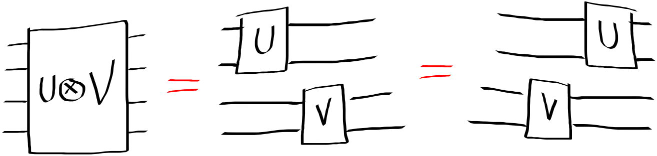
- Complete with respect to C^*-algebras.

Prior work – Staton [2015]

- Equational theory for algebra with unitaries and classical control.



- Complete with respect to C^*-algebras.
- Procedural axioms based on diagrams
  - symmetric monoidal structure



Goal

Equational theory for *embedded* quantum circuit language.

Goal

Equational theory for *embedded* quantum circuit language.

- Specialized to an embedded programming language
  - not algebra or diagrams (e.g. ZX calculus [Backens, 2015])

Goal

Equational theory for *embedded* quantum circuit language.

- Specialized to an embedded programming language
  - not algebra or diagrams (e.g. ZX calculus [Backens, 2015])
- Fewer "procedural" axioms, focus on interesting axioms.

Goal

Equational theory for *embedded* quantum circuit language.

- Specialized to an embedded programming language
  - not algebra or diagrams (e.g. ZX calculus [Backens, 2015])
- Fewer "procedural" axioms, focus on interesting axioms.
- Completeness of axioms by comparing with Staton's theory.

Homotopy type theory (HoTT): a type theory of equality

- Equality of two terms $a = b$ is a type

Homotopy type theory (HoTT): a type theory of equality

- Equality of two terms $a = b$ is a type
- Constructor: $1_a : a = a$

Homotopy type theory (HoTT): a type theory of equality

- Equality of two terms $a = b$ is a type
- Constructor: $1_a : a = a$
- Terms of equality type $p : a = b$ called *paths*

Homotopy type theory (HoTT): a type theory of equality

- Equality of two terms $a = b$ is a type
- Constructor: $1_a : a = a$
- Terms of equality type $p : a = b$ called *paths*
- Path induction:

$$\frac{H : \forall(a, b : A).\ a = b \to \mathrm{Type} \qquad \forall(a : A).\ H(1_a)}{\texttt{path\_ind}_H : \forall(a, b : A).\ \forall(p : a = b).\ H(p)}$$

Homotopy type theory (HoTT): a type theory of equality

- Equality of two terms $a = b$ is a type
- Constructor: $1_a : a = a$
- Terms of equality type $p : a = b$ called *paths*
- Path induction:

$$\frac{H : \forall(a, b : A).\ a = b \to \mathrm{Type} \qquad \forall(a : A).\ H(1_a)}{\mathtt{path\_ind}_H : \forall(a, b : A).\ \forall(p : a = b).\ H(p)}$$

- Equivalence class of an element $a : A$ with respect to a relation R: $[a]_R = [b]_R$ if $(a, b) \in R$.

Higher Inductive Type (HIT)

Definition

The quotient of a type A by a relation $R : A \to A \to \text{Prop}$ is a type A/R with data constructor:

$$\frac{a : A}{[a]_R : A/R}$$

...

Higher Inductive Type (HIT)

Definition

The quotient of a type A by a relation $R : A \to A \to \text{Prop}$ is a type A/R with data constructor:

$$\frac{a : A}{[a]_R : A/R}$$

... and path constructor:

$$\frac{a, b : A \qquad p : R(a, b)}{[p] : [a]_R = [b]_R}$$

Higher Inductive Type (HIT)

Definition

The quotient of a type A by a relation $R : A \to A \to \mathrm{Prop}$ is a type A/R with data constructor:

$$\frac{a : A}{[a]_R : A/R}$$

... and path constructor:

$$\frac{a, b : A \qquad p : R(a, b)}{[p] : [a]_R = [b]_R}$$

Note

If $p, q : R(a, b)$ and $p \neq q$, then $[p] \neq [q]$.

So what?

- HITs use paths to represent *equivalence relations* or *groupoids*.

So what?

- HITs use paths to represent *equivalence relations* or *groupoids*.
- Path induction still holds of HITs:
  - Prove theorems about groupoids by showing property holds of $1_a : a = a$.

So what?

- HITs use paths to represent *equivalence relations* or *groupoids*.
- Path induction still holds of HITs:
  - Prove theorems about groupoids by showing property holds of $1_a : a = a$.
- Unitary transformations form a groupoid.

Idea: Represent Unitaries as paths

Idea: Represent Unitaries as paths

- UMatrix(α, β) is the type of unitary matrices of dimension $|\alpha| \times |\beta|$.
  - α, β : FinType

Idea: Represent Unitaries as paths

- UMatrix(α, β) is the type of unitary matrices of dimension $|\alpha| \times |\beta|$.
  - α, β : FinType
- Quantum types: QType = FinType/UMatrix.
  - Qubit = $[\text{Bool}]_{\text{UMatrix}}$

Idea: Represent Unitaries as paths

- UMatrix(α, β) is the type of unitary matrices of dimension $|\alpha| \times |\beta|$.
  - α, β : FinType
- Quantum types: QType = FinType/UMatrix.
  - Qubit $= [\text{Bool}]_{\text{UMatrix}}$
- Unitaries are paths:

$$\frac{U : \text{UMatrix}(\alpha, \beta)}{[U] : [\alpha] = [\beta]}$$

  - $[H] : \mathit{Qubit} = \mathit{Qubit}$

HoTT QNQ calculus

$$\sigma \in \mathsf{QType} = \mathsf{FinType}/\mathsf{UMatrix}$$

$$\mathsf{Lower}\ \alpha \equiv [\alpha]_{\mathsf{UMatrix}}$$

$$\begin{aligned} e &:= x \mid \mathsf{let}\ x := e\ \mathsf{in}\ e' \\ &\mid (e_1, e_2) \mid \mathsf{let}\ (x_1, x_2) := e\ \mathsf{in}\ e' \\ &\mid \mathsf{put}\ a \mid e >!\ f \mid \cdots \end{aligned}$$

HoTT QNQ calculus

$$
\begin{aligned}
\sigma &\in \mathsf{QType} = \mathsf{FinType}/\mathsf{UMatrix} \\
\mathsf{Lower}\ \alpha &\equiv [\alpha]_{\mathsf{UMatrix}} \\
e &:= x \mid \mathsf{let}\ x := e\ \mathsf{in}\ e' \\
&\mid (e_1, e_2) \mid \mathsf{let}\ (x_1, x_2) := e\ \mathsf{in}\ e' \\
&\mid \mathsf{put}\ a \mid e >!\ f \mid \cdots
\end{aligned}
$$

- Derive $|b\rangle$ and meas e using Lower

HoTT QNQ calculus

$$\begin{aligned} \sigma &\in \mathsf{QType} = \mathsf{FinType}/\mathsf{UMatrix} \\ \mathsf{Lower}\ \alpha &\equiv [\alpha]_{\mathsf{UMatrix}} \\ e &:= x \mid \mathsf{let}\ x := e\ \mathsf{in}\ e' \\ &\mid (e_1, e_2) \mid \mathsf{let}\ (x_1, x_2) := e\ \mathsf{in}\ e' \\ &\mid \mathsf{put}\ a \mid e >!\ f \mid \cdots \end{aligned}$$

- Derive $|b\rangle$ and meas e using Lower
- Derive unitaries...

Unitaries in HoTT QNQ

Theorem

Let U be a unitary transformation $U : \sigma = \tau$.
(σ, τ : QType $\equiv$ FinType/UMatrix)

If $\Delta \vdash e : \sigma$ then there exists another expression $\Delta \vdash U \# e : \tau$.
(apply the unitary U to e)

Unitaries in HoTT QNQ

Theorem

Let U be a unitary transformation $U : \sigma = \tau$.
(σ, τ : QType $\equiv$ FinType/UMatrix)

If $\Delta \vdash e : \sigma$ then there exists another expression $\Delta \vdash U \# e : \tau$.
(apply the unitary U to e)

Proof.

By path induction. The proposition is true for $1_\sigma : \sigma = \sigma$:

$$1_\sigma \# e \equiv e$$

□

Unitaries in HoTT QNQ

Theorem

Let U be a unitary transformation $U : \sigma = \tau$.

$(\sigma, \tau : QType \equiv FinType/UMatrix)$

If $\Delta \vdash e : \sigma$ then there exists another expression $\Delta \vdash U \# e : \tau$.

(apply the unitary U to e)

Proof.

By path induction. The proposition is true for $1_\sigma : \sigma = \sigma$:

$$1_\sigma \# e \equiv e$$

□

Note

$[H] \# e \neq e$ *because* $[H] \neq 1_{Qubit}$

Unitaries in the HoTT QNQ

Theorem

Let $U : \sigma = \tau$ and $V : \tau = \rho$ be unitary transformations. Then

$$V \# (U \# e) = (V \circ U) \# e.$$

Unitaries in the HoTT QNQ

Theorem

Let $U : \sigma = \tau$ and $V : \tau = \rho$ be unitary transformations. Then

$$V \# (U \# e) = (V \circ U) \# e.$$

Proof.

By path induction on V. If $V \equiv 1_\tau$ then

$$LHS = 1_\tau \# (U \# e) = U \# e$$
$$RHS = (1_t \circ U) \# e = U \# e$$

□

We can prove a lot...

Theorem

$U^{\dagger} \# (U \# e) = e$

We can prove a lot...

Theorem

$U^{\dagger} \# (U \# e) = e$

Theorem

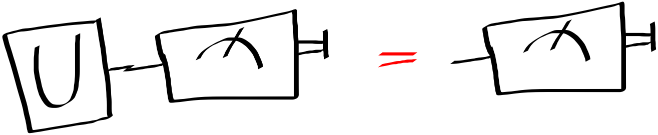
$(U_1 \otimes U_2) \# (e_1, e_2) = (U_1 \# e_1, U_2 \# e_2)$

We can prove a lot...

Theorem
$U^\dagger \# (U \# e) = e$

Theorem
$(U_1 \otimes U_2) \# (e_1, e_2) = (U_1 \# e_1, U_2 \# e_2)$

Theorem
$discard(meas(U \# e)) = discard(meas(e))$

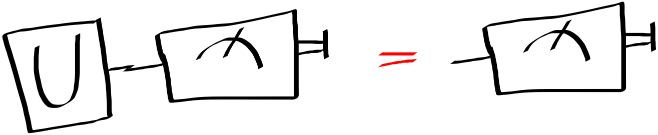


We can prove a lot...

Theorem
$U^{\dagger} \# (U \# e) = e$

Theorem
$(U_1 \otimes U_2) \# (e_1, e_2) = (U_1 \# e_1, U_2 \# e_2)$

Theorem
$discard(meas(U \# e)) = discard(meas(e))$



Theorem
$X \# |0\rangle = |1\rangle \qquad meas(X \# e) = \neg meas(e)$

...but not everything

Theorem

$SWAP \# (e_1, e_2) = (e_2, e_1)$

Proof.

???? □

...but not everything

Theorem

$SWAP \# (e_1, e_2) = (e_2, e_1)$

Proof.

???? □

Theorem

$let\ (x, y) := SWAP \# e\ in\ e' = let\ (y, x) := e\ in\ e'$

Proof.

???? □

...but not everything

Theorem
$SWAP \,\#\, (e_1, e_2) = (e_2, e_1)$

Proof.
???? □

Theorem
$let\ (x, y) := SWAP \,\#\, e\ in\ e' = let\ (y, x) := e\ in\ e'$

Proof.
???? □

Similar results for behavior of other “structural” unitaries:

$$\begin{aligned} \text{ASSOC} &: \sigma_1 \otimes (\sigma_2 \otimes \sigma_3) = (\sigma_1 \otimes \sigma_2) \otimes \sigma_3 \\ \text{LUNIT} &: () \otimes \sigma = \sigma \\ &\vdots \end{aligned}$$

Partial initialization axiom

SWAP is a structural equivalence of type $\forall X, Y.\ X \otimes Y \to Y \otimes X$ defined by the function

$$\mathsf{swap}(x, y) = (y, x)$$

Partial initialization axiom

SWAP is a structural equivalence of type $\forall X, Y.\ X \otimes Y \to Y \otimes X$ defined by the function

$$\mathsf{swap}(x, y) = (y, x)$$

Structural equivalences all correspond to unitaries

$$\widehat{\mathsf{swap}} : \forall \sigma, \tau.\ \sigma \otimes \tau = \tau \otimes \sigma$$

Partial initialization axiom

SWAP is a structural equivalence of type $\forall X, Y.\ X \otimes Y \to Y \otimes X$ defined by the function

$$\mathsf{swap}(x, y) = (y, x)$$

Structural equivalences all correspond to unitaries

$$\widehat{\mathsf{swap}} : \forall \sigma, \tau.\ \sigma \otimes \tau = \tau \otimes \sigma$$

The *partial initialization* a state $X \otimes Y$ is a pair of expressions.

$$\begin{aligned} \mathsf{init}_X\ e &\equiv e \\ \mathsf{init}_{\mathsf{Qubit}}\ (b : \mathsf{Bool}) &\equiv |b\rangle \\ \mathsf{init}_{\sigma\otimes\tau}\ (a, b) &\equiv (\mathsf{init}_\sigma\ a, \mathsf{init}_\tau\ b) \end{aligned}$$

Partial initialization axiom

SWAP is a structural equivalence of type $\forall X, Y.\ X \otimes Y \to Y \otimes X$ defined by the function

$$\mathsf{swap}(x, y) = (y, x)$$

Structural equivalences all correspond to unitaries

$$\widehat{\mathsf{swap}} : \forall \sigma, \tau.\ \sigma \otimes \tau = \tau \otimes \sigma$$

The *partial initialization* a state $X \otimes Y$ is a pair of expressions.

$$\begin{aligned} \mathsf{init}_X\ e &\equiv e \\ \mathsf{init}_{\mathsf{Qubit}}\ (b : \mathsf{Bool}) &\equiv |b\rangle \\ \mathsf{init}_{\sigma\otimes\tau}\ (a, b) &\equiv (\mathsf{init}_\sigma\ a, \mathsf{init}_\tau\ b) \end{aligned}$$

Axiom

Let f be a structural equivalence. Then

$$\widehat{f} \mathbin{\#} \mathit{init}(b) \approx \mathit{init}(f(b))$$

Partial measurement axiom

Partial measurement or *partial observation*:

$$
\begin{aligned}
\mathsf{match}_X\ e\ \mathsf{with}\ f &= \mathsf{let}\ x := e\ \mathsf{in}\ f\ x \\
\mathsf{match}_{\mathsf{Qubit}}\ e\ \mathsf{with}\ f &= e >!\ f \\
\mathsf{match}_{\sigma\otimes\tau}\ e\ \mathsf{with}\ f &= \mathsf{let}\ (x,y) := e\ \mathsf{in} \\
&\qquad\qquad \mathsf{match}_\sigma\ x\ \mathsf{with}\ (\mathsf{match}_\tau\ y\ \mathsf{with}\ f(x,y)) \\
&\ldots
\end{aligned}
$$

Partial measurement axiom

Partial measurement or *partial observation*:

$$
\begin{aligned}
\mathsf{match}_X\ e\ \mathsf{with}\ f &= \mathsf{let}\ x := e\ \mathsf{in}\ f\ x \\
\mathsf{match}_{\mathsf{Qubit}}\ e\ \mathsf{with}\ f &= e >!\ f \\
\mathsf{match}_{\sigma\otimes\tau}\ e\ \mathsf{with}\ f &= \mathsf{let}\ (x,y) := e\ \mathsf{in} \\
&\qquad\qquad \mathsf{match}_\sigma\ x\ \mathsf{with}\ (\mathsf{match}_\tau\ y\ \mathsf{with}\ f(x,y)) \\
&\ldots
\end{aligned}
$$

Axiom

Let f be a structural equivalence. Then:

$$match\ \ \widehat{f}\ \#\ e\ with\ g \approx match\ \ e\ with\ g \circ f$$

Results

- Two axioms:
  - structural unitaries + initialization
  - structural unitaries + measurement

Results

- Two axioms:
  - structural unitaries + initialization
  - structural unitaries + measurement
- Quantum programming language embedded in HoTT
  - (Finite) classical data, tuples, and sums

Results

- Two axioms:
  - structural unitaries + initialization
  - structural unitaries + measurement
- Quantum programming language embedded in HoTT
  - (Finite) classical data, tuples, and sums
- Complete with respect to Staton's equational theory

Results

- Two axioms:
  - structural unitaries + initialization
  - structural unitaries + measurement
- Quantum programming language embedded in HoTT
  - (Finite) classical data, tuples, and sums
- Complete with respect to Staton's equational theory
- Sound with respect to density matrices

Results

- Pros: theorems for free with path induction
- Cons:
  - theorems not actually free
  - no normalization
  - steep learning curve

Results

- Pros: theorems for free with path induction
- Cons:
  - theorems not actually free
  - no normalization
  - steep learning curve
- Takeaway: Equations stem (mostly) from quantum data/classical control, not artificial axioms

Results

- Pros: theorems for free with path induction
- Cons:
  - theorems not actually free
  - no normalization
  - steep learning curve
- Takeaway: Equations stem (mostly) from quantum data/classical control, not artificial axioms

Thanks!

A HoTT Quantum Equational Theory

Jennifer Paykin
Galois, Inc
jpaykin@galois.com

MURI Project Review
University of Maryland

March 8, 2019

Questions?

Supported by FA9550-16-1-0082
Semantics and Structures for Higher-level Quantum Programming Languages

References I

M. Backens. *Completeness and the ZX-calculus*. PhD thesis, University of Oxford, 02 2015.

N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135. Springer Berlin Heidelberg, 1995. doi: 10.1007/BFb0022251.

A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 333–342, New York, NY, USA, 2013. ACM. doi: 10.1145/2491956.2462177.

References II

J. Paykin, R. Rand, and S. Zdancewic. QWIRE: A core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 846–858, New York, NY, USA, 2017. ACM. doi: 10.1145/3009837.3009894.

R. Rand, J. Paykin, and S. Zdancewic. QWIRE practice: Formal verification of quantum circuits in Coq. In *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, The Netherlands, 3-7 July 2017.*, pages 119–132, 2017. doi: 10.4204/EPTCS.266.8.

S. Staton. Algebraic effects, linearity, and quantum programming languages. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 395–406, New York, NY, USA, 2015. ACM. doi: 10.1145/2676726.2676999.