Towards a monadic semantics of quantum computation

Thorsten Altenkirch
University of Nottingham
Is quantum computation possible?

Yes
We will be able to exploit quantum parallelism to speed up computations.
Research in quantum computing is justified.

No
Quantum parallelism does not occur or cannot be scaled up.
Nature behaves classically, computationally.
We should be able to simulate physical systems efficiently on a classical computer.
Research in quantum computing is justified.
Is quantum computation possible?

Yes: We will be able to exploit quantum parallelism to speed up computations.
Is quantum computation possible?

Yes

- We will be able to exploit quantum parallelism to speed up computations.
- Research in quantum computing is justified.

No

- Quantum parallelism does not occur or cannot be scaled up.
- Nature behaves classically, computationally.
- We should be able to simulate physical systems efficiently on a classical computer.
- Research in quantum computing is justified.
Is quantum computation possible?

Yes  We will be able to exploit quantum parallelism to speed up computations. Research in quantum computing is justified.

No  Quantum parallelism does not occur or cannot be scaled up.
Is quantum computation possible?

Yes  
- We will be able to exploit quantum parallelism to speed up computations.
- Research in quantum computing is justified.

No  
- Quantum parallelism does not occur or cannot be scaled up.
- Nature behaves classically, computationally.
Is quantum computation possible?

**Yes**
- We will be able to exploit quantum parallelism to speed up computations.
- Research in quantum computing is justified.

**No**
- Quantum parallelism does not occur or cannot be scaled up.
- Nature behaves classically, computationally.
- We should be able to simulate physical systems efficiently on a classical computer.
Is quantum computation possible?

Yes
- We will be able to exploit quantum parallelism to speed up computations.
- Research in quantum computing is justified.

No
- Quantum parallelism does not occur or cannot be scaled up.
- Nature behaves classically, computationally.
- We should be able to simulate physical systems efficiently on a classical computer.
- Research in quantum computing is justified.
The goal

To develop a framework where we can express and combine irreversible quantum effects and conventional algorithms.
The goal

To develop a framework where we can express and combine irreversible quantum effects and conventional algorithms. We follow the idea of monadic effects introduced by Eugenio Moggi to structure denotational semantics.
The goal

To develop a framework where we can express and combine irreversible quantum effects and conventional algorithms. We follow the idea of monadic effects introduced by Eugenio Moggi to structure denotational semantics.

- introduced by Eugenio Moggi to structure denotational semantics.

- popularized by Phil Wadler as a means to introduce effects in Haskell and to structure functional programs.
What is a (computational) monad?
What is a (computational) monad?

An operator $T$ on objects

$$\frac{A \in C}{T(A) \in C}$$
What is a (computational) monad?

An operator $T$ on objects

$$\frac{A \in C}{T(A) \in C}$$

$T(A)$ computations over $A$
What is a (computational) monad?

An operator $T$ on objects

\[
\frac{A \in C}{T(A) \in C}
\]

$T(A)$ computations over $A$

unit

\[
\frac{A \in C}{\eta_A \in C(A, T(A))}
\]
What is a (computational) monad?

An operator $T$ on objects

$$A \in C \quad \Rightarrow \quad T(A) \in C$$

$T(A)$ computations over $A$

**unit**

$$A \in C \quad \Rightarrow \quad \eta_A \in \text{C}(A, T(A))$$

**bind**

$$f \in \text{C}(A, T(B)) \quad \Rightarrow \quad \hat{f} \in \text{C}(T(A), T(B))$$
What is a computational monad?

Equations

\[ \hat{\eta}_A = 1_A \]

\[ \hat{f} \circ \eta_A = f \]

\[ \hat{g} \circ f = \hat{g} \circ \hat{f} \]
Remarks
Remarks

\( (T, \eta, \hat{\cdot}) \) is a Kleisli triple.
Remarks

- $(T, \eta, \hat{-})$ is a Kleisli triple.

- Equivalent to the usual presentation of monads using using a functor $T$ and $\mu_A : T(T(A)) \to T(A)$. 
(\(T, \eta, \hat{-}\)) is a Kleisli triple.

Equivalent to the usual presentation of monads using using a functor \(T\) and \(\mu_A : T(T(A)) \to T(A)\).

Monads in Haskell use

\[\text{bind}_{A,B} \in T(A) \to (A \to T(B)) \to T(B)\]
Example: the state monad
Example: the state monad

Given a type of states

$$S_t \in \text{Set}$$

we define a monad $S$. 
Example: the state monad

Given a type of states

\[ S_t \in \text{Set} \]

we define a monad \( S \).

\[ S(A) \in \text{Set} \]

\[ S(A) = S_t \to A \times S_t \]
Example: the state monad

\[ \eta_A \in A \rightarrow S(A) \]
\[ \eta_A(a) = \lambda s.(a, s) \]
Example: the state monad

\[ \eta_A \in A \to S(A) \]
\[ \eta_A(a) = \lambda s. (a, s) \]

\[ f \in A \to S(B) \]
\[ \hat{f} : S(A) \to S(B) \]
\[ \hat{f}(\sigma) = \lambda s : S. f(a)(s') \]
\[ \text{where } (a, s') = \sigma(s) \]
Operations on $S$

\[
\text{set} \in St \rightarrow S(1) \\
\text{set}(s) = \lambda s'.((), s) \\
\text{get} \in 1 \rightarrow S(St) \\
\text{get}() = \lambda s.(s, s)
\]
The Kleisli category

Objects

Morphisms

Identity

Composition

Equations follow from monadic equations.

Towards a monadic semantics of quantum computation – p.10/
The Kleisli category

Given a monad \( T \in \mathbb{C} \rightarrow \mathbb{C} \) we define the *Kleisli category* \( \mathbb{C}_T \) as
The Kleisli category

Given a monad $T \in C \to C$ we define the *Kleisli category* $C_T$ as

**Objects** Objects of $C$
The Kleisli category

Given a monad $T \in C \to C$ we define the Kleisli category $C_T$ as

**Objects**  Objects of $C$

**Morphisms**  $C_T(A, B) = C(A, T(B))$
The Kleisli category

Given a monad $T \in C \to C$ we define the *Kleisli category* $C_T$ as

**Objects** Objects of $C$

**Morphisms** $C_T(A, B) = C(A, T(B))$

**Identity** $1_A = \eta_A \in C_T(A, A)$
The Kleisli category

Given a monad $T \in \mathcal{C} \to \mathcal{C}$ we define the Kleisli category $\mathcal{C}_T$ as

**Objects** Objects of $\mathcal{C}$

**Morphisms** $\mathcal{C}_T(A, B) = \mathcal{C}(A, T(B))$

**Identity** $1_A = \eta_A \in \mathcal{C}_T(A, A)$

**Composition** $g \in \mathcal{C}(A, T(B)), f \in \mathcal{C}(B, T(C))$

$f \ast g = \hat{f} \circ g$
The Kleisli category

Given a monad \( T \in \mathcal{C} \to \mathcal{C} \), we define the Kleisli category \( \mathcal{C}_T \) as:

- **Objects**: Objects of \( \mathcal{C} \)
- **Morphisms**: \( \mathcal{C}_T(A, B) = \mathcal{C}(A, T(B)) \)
- **Identity**: \( 1_A = \eta_A \in \mathcal{C}_T(A, A) \)
- **Composition**: \( g \in \mathcal{C}(A, T(B)), f \in \mathcal{C}(B, T(C)) \)
  \[ f \circ g = \hat{f} \circ g \]

Equations follow from monadic equations.
In the case of $S$ we have

$$\text{Set}_S(A, B) \simeq A \times St \rightarrow B \times St$$

$$\text{set} \in \text{Set}_S(\text{St}, 1)$$

$$\text{get} \in \text{Set}_S(1, \text{St})$$
Observations

gives a denotational semantics for computations with state. We can also implement operationally by using real side effects. In the case of there isn’t a huge difference between both views. Haskell uses both views of monads denotational Maybe, [], operational IO.
Observations

- $S$ gives a *denotational semantics* for computations with state.
$S$ gives a *denotational semantics* for computations with state.

We can also *implement* $S$ operationally by using real side effects.
Observations

- $S$ gives a *denotational semantics* for computations with state.
- We can also *implement* $S$ operationally by using real side effects.
- In the case of $S$ there isn’t a huge difference between both views.
Observations

- $S$ gives a *denotational semantics* for computations with state.
- We can also *implement* $S$ operationally by using real side effects.
- In the case of $S$ there isn’t a huge difference between both views.
- Haskell uses both views of monads
Observations

$S$ gives a denotational semantics for computations with state.

We can also implement $S$ operationally by using real side effects.

In the case of $S$ there isn’t a huge difference between both views.

Haskell uses both views of monads 

denotational Maybe, [] ...
Observations

- $S$ gives a *denotational semantics* for computations with state.
- We can also *implement* $S$ operationally by using real side effects.
- In the case of $S$ there isn’t a huge difference between both views.
- Haskell uses both views of monads
  - denotational `Maybe`, `[]` ...
  - operational `IO`
Probabilistic computations
Probabilistic computations

\[ P(A) = \{ \nu \in A \rightarrow \mathbb{R}^+ \mid \sum_{a \in A} \nu(a) \leq 1 \} \]
Probabilistic computations

\[
P(A) = \{ \nu \in A \rightarrow \mathbb{R}^+ \mid \sum_{a \in A} \nu(a) \leq 1 \}\]

\[
\eta_A \in A \rightarrow P(A)
\]

\[
\eta_A(a) = \lambda b . \delta_a(b)
\]
Probabilistic computations

\[
A \in \text{Set}
\]

\[
P(A) = \{ v \in A \rightarrow \mathbb{R}^+ \mid \sum_{a \in A} v(a) \leq 1 \}
\]

\[
\eta_A \in A \rightarrow P(A)
\]

\[
\eta_A(a) = \lambda b. \delta_a(b)
\]

\[
f : A \rightarrow P(B)
\]

\[
\hat{f} \in P(A) \rightarrow P(B)
\]

\[
\hat{f}(v) = \lambda b \in B. \sum_{a \in A} v(a). f(a, b)
\]
Problem
Problem

\[ \sum_{a \in A} \ldots \text{is not defined in general.} \]
Problem

\[ \sum_{a \in A} \ldots \text{ is not defined in general.} \]

For the moment we restrict ourselves to finite sets \( A \).

\[ P \in \text{Set}_{\omega} \rightarrow \text{Set} \]
Problem

\[ \sum_{a \in A} \ldots \] is not defined in general.

For the moment we restrict ourselves to finite sets \( A \).

\[ P \in \text{Set}_{\omega} \rightarrow \text{Set} \]

This doesn’t fit into the structure of a monad but is a **Kleisli structure** with \( \text{Set}_{\omega} \subseteq \text{Set} \).
Kleisli structures

Operators on Objects \( T \subseteq C \subseteq D \)

\[
\frac{A \in C}{T(A) \in D}
\]

unit and bind

\[
\frac{A \in C}{\eta_A \in D(A, T(A))}
\]

\[
\frac{f \in D(A, T(B))}{\hat{f} \in D(T(A), T(B))}
\]

Equations as before
Lifting $P$

We can lift $P$ to an operator on Sets:

$$\tilde{P} \in \text{Set} \rightarrow \text{Set}$$

$$\tilde{P}(A) = \{ v \in A \twoheadleftarrow_{<\omega} \mathbb{R}^+ | \sum_{a \in \text{dom}(v)} v(a) \leq 1 \}$$

Here $A \twoheadleftarrow_{<\omega} B$ is the set of partial functions with finite support.
Observations

\( \eta, \hat{f} \) can be extended to \( \hat{P} \).
Observations

- $\eta, \hat{f}$ can be extended to $\tilde{P}$.
- $\tilde{P}$ is a monad on $\mathbb{Set}$. 
Observations

- $\eta, \hat{f}$ can be extended to $\tilde{P}$.
- $\tilde{P}$ is a monad on $\text{Set}$.
- $\tilde{P}$ is the left Kan extension of $P$ along $I$. 
Tossing a coin

\[\text{coin} \in 1 \rightarrow P(\text{Bool})\]

\[\text{coin}() = \lambda b \in \text{Bool}. \frac{1}{2}\]

\[\text{coin} \in \text{Set}_P(1, \text{Bool})\]
Pure Quantum computations
Pure Quantum computations

\[ Q(A) = \left\{ v \in A \rightarrow \mathbb{C} \mid \sum_{a \in A} |v(a)|^2 \leq 1 \right\} \]
Pure Quantum computations

\[ Q(A) = \{ v \in A \rightarrow \mathbb{C} \mid \sum_{a \in A} |v(a)|^2 \leq 1 \} \]

\[ \eta, \hat{\cdot} \text{ as for } P. \]
Hadamard transformation

\[ H \in \text{Set}(\text{Bool}, \mathcal{Q}(\text{Bool})) \]
\[ \in \text{Set}_Q(\text{Bool}, \text{Bool}) \]
\[ H(0) = \lambda b. \sqrt{2} \]
\[ H(1) = \lambda b. \text{if } b \text{ then } -\sqrt{2} \text{ else } \sqrt{2} \]
Observations
Observations

Composition in $\text{Set}_P, \text{Set}_Q$ is matrix multiplication.
Observations

- Composition in $\text{Set}_P, \text{Set}_Q$ is matrix multiplication.
- Coproducts and products in $\text{Set}$ induce monoidal connectives in $\text{Set}_P, \text{Set}_Q$
Observations

- Composition in $\text{Set}_P, \text{Set}_Q$ is matrix multiplication.
- Coproducts and products in $\text{Set}$ induce monoidal connectives in $\text{Set}_P, \text{Set}_Q$
  \[ A \oplus_{\text{Set}_P,Q} B = A + B \]
  Cartesian product of vectors
Observations

- Composition in $\mathbf{Set}_P, \mathbf{Set}_Q$ is matrix multiplication.

- Coproducts and products in $\mathbf{Set}$ induce monoidal connectives in $\mathbf{Set}_P, \mathbf{Set}_Q$
  
  \[
  A \oplus_{\mathbf{Set}_P,Q} B = A + B
  \]
  Cartesian product of vectors
  
  \[
  A \otimes_{\mathbf{Set}_P,Q} B = A \times B
  \]
  Tensor product

Towards a monadic semantics of quantum computation – p.21/?
Observations

- Composition in $\text{Set}_P, \text{Set}_Q$ is matrix multiplication.
- Coproducts and products in $\text{Set}$ induce monoidal connectives in $\text{Set}_P, \text{Set}_Q$
  \[ A \oplus_{\text{Set}_{P,Q}} B = A + B \]
  Cartesian product of vectors
  \[ A \otimes_{\text{Set}_{P,Q}} B = A \times B \]
  Tensor product
- The denotational complexity of $\text{Set}_P, \text{Set}_Q$ is the same.
Observations

- Operationally $\mathsf{Set}_P$ can be easily realized.
Observations

- Operationally $\text{Set}_P$ can be easily realized.
- $\text{Set}_Q$ includes quantum algorithms and seems to have no efficient classical implementation.
Observations

- Operationally $\text{Set}_P$ can be easily realized.
- $\text{Set}_Q$ includes quantum algorithms and seems to have no efficient classical implementation.
- Morphisms in $\text{Set}_Q$ are arbitrary matrices, not only unitary ones.
Observations

- Operationally $\text{Set}_P$ can be easily realized.
- $\text{Set}_Q$ includes quantum algorithms and seems to have no efficient classical implementation.
- Morphisms in $\text{Set}_Q$ are arbitrary matrices, not only unitary ones.
- We want to model *irreversible* quantum computations.
Observations

- Operationally $\text{Set}_P$ can be easily realized.
- $\text{Set}_Q$ includes quantum algorithms and seems to have no efficient classical implementation.
- Morphisms in $\text{Set}_Q$ are arbitrary matrices, not only unitary ones.
- We want to model *irreversible* quantum computations.
- However, irreversible steps (measurements) lead to mixed states - this is not modelled by $\text{Set}_Q$. 

Towards a monadic semanticsof quantum computation – p.22?
Mixed states as a monad?
Mixed states as a monad?

Mixed states as probability distributions over pure states
Mixed states as a monad?

Mixed states as probability distributions over pure states

\[ P_Q(A) \in \text{Set}_{<\omega} \rightarrow \text{Set} \]
\[ = \tilde{P}(Q(A)) \]
\[ = \{ f \in Q(A) \rightarrow_{<\omega} \mathbb{R}^+ \mid \Sigma v \in \text{dom}(f).f(v) \leq 1 \} \]
Density matrices

We can represent mixed states as density matrices:
Density matrices

\[ D(A) = \{ f \in A \times A \to \mathbb{C} \mid \text{tr}(f) \leq 1 \land f \text{ positive hermitian} \} \]
Density matrices

\[ D(A) = \{ f \in A \times A \rightarrow \mathbb{C} \mid \text{tr}(f) \leq 1 \wedge \text{f positive hermitian} \} \]

We can represent mixed states as density matrices:

\[ \Phi \in PQ(A) \rightarrow D(A) \]

\[ \Phi(\nu) = \lambda(a, b). \sum_{w \in \text{dom}(\nu)} \nu(w)w(a)w(a)^* \]
Partial superoperators

Morphisms between density matrices are superoperators (completely positive, non-trace-increasing operators).
Partial superoperators

Morphisms between density matrices are superoperators (completely positive, non-trace-increasing operators). Can we find a monadic representation of this category?