Towards a monadic semantics of quantum computation

Thorsten Altenkirch

University of Nottingham

Is quantum computation possible?

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.

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 effciently 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 effciently 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 coventional algorithms.

The goal

To develop a framework where we can express and combine irreversible quantum effects and coventional 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 coventional algorithms.
We follow the idea of monadic effects

- 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

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

What is a (computational) monad?

An operator T on objects

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

$T(A)$ computations over A

What is a (computational) monad?

An operator T on objects

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

$T(A)$ computations over A

unit $$\frac{A \in \mathbf{C}}{\eta_A \in \mathbf{C}(A, T(A))}$$

What is a (computational) monad?

An operator T on objects

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

$T(A)$ computations over A

unit $$\frac{A \in \mathbf{C}}{\eta_A \in \mathbf{C}(A, T(A))}$$

bind $$\frac{f \in \mathbf{C}(A, T(B))}{\hat{f} \in \mathbf{C}(T(A), T(B))}$$

What is a computational monad?

Equations

$$
\begin{aligned}
\hat{\eta}_A &= 1_A \\
\hat{f} \circ \eta_A &= f \\
\widehat{\hat{g} \circ f} &= \hat{g} \circ \hat{f}
\end{aligned}
$$

Remarks

Remarks

- $(T, \eta, \hat{-})$ 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 $\mu_A : T(T(A)) \to T(A)$.

Remarks

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

$$\mathtt{bind}_{A,B} \in T(A) \to (A \to T(B)) \to T(B)$$

Example: the state monad

Example: the state monad

Given a type of states

$$\mathrm{St} \in \mathbf{Set}$$

we define a monad S.

Example: the state monad

Given a type of states

$$\mathrm{St} \in \mathbf{Set}$$

we define a monad S.

$$\frac{A \in \mathbf{Set}}{\begin{array}{rcl} S(A) & \in & \mathbf{Set} \\ S(A) & = & \mathrm{St} \to A \times \mathrm{St} \end{array}}$$

Example: the state monad

$$\begin{aligned} \eta_A &\in A \to S(A) \\ \eta_A(a) &= \lambda s.(a, s) \end{aligned}$$

Example: the state monad

$$\begin{aligned} \eta_A &\in A \to S(A) \\ \eta_A(a) &= \lambda s.(a, s) \end{aligned}$$

$$\frac{f \in A \to S(B)}{\begin{aligned} \hat{f} &: S(A) \to S(B) \\ \hat{f}(\sigma) &= \lambda s : \mathrm{St}. f(a)(s') \\ &\quad \text{where } (a, s') = \sigma(s) \end{aligned}}$$

Operations on S

$$
\begin{aligned}
\text{set} &\in \text{St} \to S(1) \\
\text{set}(s) &= \lambda s'.((), s)
\end{aligned}
$$

$$
\begin{aligned}
\text{get} &\in 1 \to S(\text{St}) \\
\text{get}() &= \lambda s.(s, s)
\end{aligned}
$$

The Kleisli category

The Kleisli category

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

The Kleisli category

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

Objects Objects of $\mathbf{C}$

The Kleisli category

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

Objects Objects of $\mathbf{C}$

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

The Kleisli category

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

Objects Objects of $\mathbf{C}$

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

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

The Kleisli category

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

Objects Objects of $\mathbf{C}$

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

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

Composition $$\frac{g \in \mathbf{C}(A, T(B)), f \in \mathbf{C}(B, T(C))}{f * g = \hat{f} \circ g}$$

The Kleisli category

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

Objects Objects of $\mathbf{C}$

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

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

Composition $$\frac{g \in \mathbf{C}(A, T(B)), f \in \mathbf{C}(B, T(C))}{f * g = \hat{f} \circ g}$$

Equations follow from monadic equations.

$\mathbf{Set}_S$

In the case of S we have

$$\mathbf{Set}_S(A, B) \simeq A \times St \to B \times St$$

$$\begin{aligned} \mathrm{set} &\in \mathbf{Set}_S(\mathrm{St}, 1) \\ \mathrm{get} &\in \mathbf{Set}_S(1, \mathrm{St}) \end{aligned}$$

Observations

Observations

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

Observations

- 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

$$\frac{A \in \mathbf{Set}}{P(A) = \{v \in A \to \mathbb{R}^+ \mid \Sigma_{a \in A}.v(a) \leq 1\}}$$

Probabilistic computations

$$\frac{A \in \mathbf{Set}}{P(A) = \{v \in A \to \mathbb{R}^+ \mid \Sigma_{a \in A}.v(a) \leq 1\}}$$

$$\begin{aligned} \eta_A &\in A \to P(A) \\ \eta_A(a) &= \lambda b.\delta_a(b) \end{aligned}$$

Probabilistic computations

$$\frac{A \in \mathbf{Set}}{P(A) = \{v \in A \to \mathbb{R}^+ \mid \Sigma_{a \in A}.v(a) \leq 1\}}$$

$$\begin{aligned} \eta_A &\in A \to P(A) \\ \eta_A(a) &= \lambda b.\delta_a(b) \end{aligned}$$

$$\frac{f : A \to P(B)}{\begin{aligned} \hat{f} &\in P(A) \to P(B) \\ \hat{f}(v) &= \lambda b \in B.\Sigma_{a \in A} v(a) f(a,b) \end{aligned}}$$

Problem

Problem

- $\sum_{a \in A} \ldots$ is not defined in general.

Problem

- $\Sigma_{a \in A} \ldots$ is not defined in general.
- For the moment we restrict ourselves to finite sets A.

$$P \in \mathbf{Set}_{<\omega} \to \mathbf{Set}$$

Problem

- $\Sigma_{a\in A}\ldots$ is not defined in general.
- For the moment we restrict ourselves to finite sets A.

$$P \in \mathbf{Set}_{<\omega} \to \mathbf{Set}$$

- This doesn't fit into the structure of a monad but is a *Kleisli structure* with $\mathbf{Set}_{<\omega} \subseteq \mathbf{Set}$.

Kleisli structures

Operators on Objects $T \ \mathbf{C} \subseteq \mathbf{D}$

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

unit and bind $\dfrac{A \in \mathbf{C}}{\eta_A \in \mathbf{D}(A, T(A))}$

$$\frac{f \in \mathbf{D}(A, T(B))}{\hat{f} \in \mathbf{D}(T(A), T(B))}$$

Equations *as before*

Lifting P

We can lift P to an operator on Sets:

$$\begin{array}{rcl} \tilde{P} & \in & \mathbf{Set} \to \mathbf{Set} \\ \tilde{P}(A) & = & \{v \in A \rightharpoonup_{<\omega} \mathbb{R}^+ \mid \Sigma_{a \in \mathrm{dom}(v)} v(a) \leq 1\} \end{array}$$

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

Observations

Observations

- $\eta, \hat{f}$ can be extended to $\tilde{P}$.

Observations

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

Observations

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

Tossing a coin

$$\begin{aligned} \text{coin} &\in 1 \to P(\text{Bool}) \\ \text{coin}() &= \lambda b \in \text{Bool}.\frac{1}{2} \end{aligned}$$

$$\text{coin} \in \mathbf{Set}_P(1, \text{Bool})$$

Pure Quantum computations

Pure Quantum computations

$$\frac{A \in \mathbf{Set}}{Q(A) = \{v \in A \to \mathbb{C} \mid \Sigma_{a \in A}.|v(a)|^2 \leq 1\}}$$

Pure Quantum computations

$$\frac{A \in \mathbf{Set}}{Q(A) = \{v \in A \to \mathbb{C} \mid \Sigma_{a \in A}.|v(a)|^2 \leq 1\}}$$

$\eta, \hat{-}$ as for P.

Hadamard transformation

$$\begin{aligned} H &\in \mathbf{Set}(\mathrm{Bool}, Q(\mathrm{Bool})) \\ &\in \mathbf{Set}_Q(\mathrm{Bool}, \mathrm{Bool}) \\ H(0) &= \lambda b.\sqrt{2} \\ H(1) &= \lambda b.\mathrm{if}\ b\ \mathtt{then}\ -\sqrt{2}\ \mathrm{else}\ \sqrt{2} \end{aligned}$$

Observations

Observations

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

Observations

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

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

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
  - $A \otimes_{\mathbf{Set}_{P,Q}} B = A \times B$
    Tensorproduct

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
  - $A \otimes_{\mathbf{Set}_{P,Q}} B = A \times B$
    Tensorproduct
- The denotational complexity of $\mathbf{Set}_P, \mathbf{Set}_Q$ is the same.

Observations

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

Observations

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

Observations

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

Observations

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

Observations

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

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

$$\begin{aligned} PQ(A) &\in \mathbf{Set}_{<\omega} \to \mathbf{Set} \\ &= \tilde{P}(Q(A)) \\ &= \{f \in Q(A) \rightharpoonup_{<\omega} \mathbb{R}^+ \mid \Sigma v \in \mathrm{dom}(f).f(v) \leq 1\} \end{aligned}$$

Density matrices

Density matrices

$$D(A) = \{f \in A \times A \to \mathbb{C} \mid \mathrm{tr}(f) \leq 1 \wedge f \text{ positive hermitian}\}$$

Density matrices

$$
\begin{aligned}
D(A) \; &= \; \{ f \in A \times A \to \mathbb{C} \mid \mathrm{tr}(f) \leq 1 \wedge \\
&\qquad f \text{ positive hermitian} \}
\end{aligned}
$$

We can represent mixed states as density matrices:

$$
\begin{aligned}
\Phi \; &\in \; PQ(A) \to D(A) \\
\Phi(v) \; &= \; \lambda(a,b).\Sigma_{w \in \mathrm{dom}(v)} v(w) w(a) w(a)^{*}
\end{aligned}
$$

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?