Functional Quantum Programming

Thorsten Altenkirch

School of Computer Science and IT
University of Nottingham

April 26, 2007

People at Nottingham

Jonathan Grattage
: recently successfully defended his PhD on QML
A Quantum Programming Language

Alex Green
: works on quantum programming in a functional setting.

Slava Belavkin
: Prof in Mathematical Physics,
Quantum Information Theory

Where do we come from . . .

- Functional Programming
  e.g. functional treatment of concurrency
- Type Theory
  e.g. Epigram: program+specify+prove
- Category Theory
  e.g. Containers for generic programming
- Quantum Programming
  e.g. QML, QIO monad

What is functional programming? And why?

- Based on function abstraction and application (λ calculus).
- Ease of building abstractions, reasoning about programs
- Programming $\sim$ constructive mathematics.
- Popular functional language: Haskell

Functional Quantum Programming?

- QML - a quantum programming language
  - Design ideas
  - Operational and denotational semantics
  - An algebra of quantum programs
- The QIO monad in Haskell
- Questions for QICS

QPL, . . .

- Starting point: Selinger's QPL
  or Simon Perdrix's quantum programming language.
- Simple language with a nice mathematical semantics (Superoperators)
- Unitary operators are represented as combinatorical expressions
  built up from some primitives, e.g. Hadmard, CNOT, etc
- Slogan: Quantum data — classical control.
- Quantum variables can only be used in a linear fashion (no contraction).

QML

- Contraction by sharing
- Explicit weakening by measurement
- Reversible **if**$^\circ$ and irreversible **if**
- Quantum data and control.
- No while loops.

Contraction by sharing

$$\delta \in Q_2 \multimap Q_2 \otimes Q_2$$
$$\delta\ x = (x, x)$$

$$\delta\ (\mathit{false} +_Q \mathit{true}) \not\equiv (\mathit{false} +_Q \mathit{true}, \mathit{false} +_Q \mathit{true})$$
$$\delta\ (\mathit{false} +_Q \mathit{true}) \equiv (\mathit{false}, \mathit{false}) +_Q (\mathit{true}, \mathit{true})$$

Explicit weakening by measurement

$$\pi_1 \in Q_2 \otimes Q_2 \multimap Q_2$$
$$\pi_1\ (x, y) = x \uparrow \{y\}$$

$$\pi_1\ (\delta\ x) \equiv x?$$
$$\pi_1\ (\delta\ (\mathit{false} +_Q \mathit{true})) \equiv \mathit{false} +_P \mathit{true}$$

Reversible $\mathbf{if}^\circ$ and irreversible $\mathbf{if}$

$$\neg \in Q_2 \multimap Q_2$$

$$\neg\ x = \mathbf{if}^\circ\ x\ \mathbf{then}\ \mathit{false}\ \mathbf{else}\ \mathit{true}$$

$$\neg\ (\neg\ x) \equiv x$$

$$\neg^c \in Q_2 \multimap Q_2$$

$$\neg^c\ x = \mathbf{if}\ x\ \mathbf{then}\ \mathit{false}\ \mathbf{else}\ \mathit{true}$$

$$\neg^c\ (\neg^c\ (\mathit{false} +_Q \mathit{true})) \equiv \mathit{false} +_P \mathit{true}$$

Why do we need **if**?

$$cswap \in Q_2 \multimap Q_2 \otimes Q_2 \multimap Q_2 \otimes Q_2$$
$$cswap\ x\ (y, z) = \mathbf{if}^{\circ}\ x\ \mathbf{then}\ (z, y)\ \mathbf{else}\ (y, z)$$

is **not well-typed**, because we cannot show $(z, y) \perp (y, z)$.

$$cswap' \in Q_2 \multimap Q_2 \otimes Q_2 \multimap Q_2 \otimes Q_2$$
$$cswap'\ x\ (y, z) = \mathbf{if}\ x\ \mathbf{then}\ (z, y)\ \mathbf{else}\ (y, z)$$

is well-typed, since **if** does not require orthogonality.

QML's type system

We introduce the following judgements:

Programs

$$\Gamma \vdash t : \sigma$$

Pure programs

$$\Gamma \vdash^{\circ} t : \sigma$$

programs without weakening and **if**.

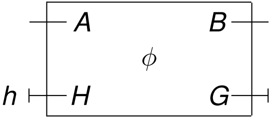
Orthogonality

$$t \perp u$$

given that $\Gamma \vdash^{\circ} t, u : \sigma$.

QML's operational semantics

Given (non-empty) finite sets A (input) and B (output), we define $\mathrm{FQC}(A, B)$ as:



- a finite set H, the base of the space of initial heaps,
- a heap initialisation vector $\vec{h} \in \mathbb{C}^H$,
- a finite set G, the base of the space of garbage states,
- a unitary operator $\phi \in A \otimes H \multimap_{\mathrm{unitary}} B \otimes G$.

QML's operational semantics

We write:

$\mathrm{FQC}(A, B)$ quantum circuits with heap and garbage.

$\mathrm{FQC}^{\circ}(A, B)$ quantum circuits with heap but no garbage.

$$\frac{\Gamma \vdash t : \sigma}{[\![t]\!]^{\mathrm{op}} \in \mathrm{FQC}([\![\Gamma]\!], [\![\sigma]\!])}$$

$$\frac{\Gamma \vdash^{\circ} t : \sigma}{[\![t]\!]^{\mathrm{op}} \in \mathrm{FQC}^{\circ}([\![\Gamma]\!], [\![\sigma]\!])}$$

QML's denotational semantics

We write:

$\mathrm{Super}(A, B)$ Superoperators

$\mathrm{Isom}(A, B)$ Isometries

$$\frac{\Gamma \vdash t : \sigma}{[\![t]\!]^{\mathrm{den}} \in \mathrm{Super}([\![\Gamma]\!], [\![\sigma]\!])}$$

$$\frac{\Gamma \vdash^{\circ} t : \sigma}{[\![t]\!]^{\mathrm{den}} \in \mathrm{Isom}([\![\Gamma]\!], [\![\sigma]\!])}$$

Relating operational and denotational semantics

We can assign denotations to circuits:

$$\frac{c \in \mathrm{FQC}^{\circ}(A, B)}{\mathrm{D}(c) \in \mathrm{Isom}(A, B)}$$

$$\frac{c \in \mathrm{FQC}(A, B)}{\mathrm{D}(c) \in \mathrm{Super}(A, B)}$$

and state soundness of the operational semantics:

$$\mathrm{D}(\llbracket t \rrbracket^{\mathrm{op}}) = \llbracket t \rrbracket^{\mathrm{den}}$$

for $\Gamma \vdash t : \sigma$ ($\Gamma \vdash^{\circ} t : \sigma$).

QUICS questions

- Can we extend QML by classical coproducts (corresponding to biproducts)?
- Can we extend QML by quantum views (corresponding to change of base)?
- Can we extend QML by higher order types?

Higher Order ?

- It may seem that higher order is no problem because in every compact closed category:

$$\mathrm{C}(A \otimes B, C) \simeq \mathrm{C}(A, B^* \otimes C)$$

- But is this the right structure?
- Superoperators are not compact closed (but completely positive maps are).
- The category of relations

$$\mathrm{Rel}(A, B) = A \to \mathcal{P}(B)$$

is compact closed, but

$$\mathrm{Rel}_{<\omega}(A, B) = A \to \mathcal{P}_{<\omega}(B)$$

is not.

Day's construction

- For any category C the category of presheaves $\mathrm{PSh}(\mathrm{C})$ is given by:

Objects	Contravariant functors from C to Set.
Morphisms	Natural transformations.

- There is an embedding Y (the Yoneda embedding) from C to $\mathrm{PSh}(\mathrm{C})$

$$\mathrm{Y}(A) = \mathrm{C}(-, A)$$

- A monoidal structure on C induces a monoidal structure in $\mathrm{PSh}(C)$:

$$(F \otimes G)(X) = \int^{A,B} F(A) \times G(B) \times \mathrm{C}(X, A \otimes B)$$

- Y preserves the monoidal structure.

Day's construction

- This structure is always closed:

$$(F \multimap G)(X) = \mathrm{Nat}(\mathrm{Y}(X) \otimes F, G)$$

- This is semantically the interpretations of higher order computations as chunks (delayed computations).

An algebra of quantum programs ?

joint work with Amr Sabry and Juliana Vizotto.

- QPL 2005: restricted to the pure fragment
  (no weakening, no **if**)
  Denotational semantics: isometries
- Extends the rules for the classical sublanguage
  (no superpositions, **if** and **if**$^\circ$ behave the same)
  Denotational semantics: sets and (injective) functions
- Sound and complete.
- Completeness also gives rise to a normalisation algorithm
  (*Normalisation by evaluation*).

Equations for $\mathbf{if}^\circ$

β

$$\begin{aligned}\mathbf{if}^\circ\ \mathit{false}\ \mathbf{then}\ t\ \mathbf{else}\ u &\equiv u\\ \mathbf{if}^\circ\ \mathit{true}\ \mathbf{then}\ t\ \mathbf{else}\ u &\equiv t\end{aligned}$$

η

$$\mathbf{if}^\circ\ t\ \mathbf{then}\ \mathit{true}\ \mathbf{else}\ \mathit{false} \equiv t$$

Commuting conversion

$$\begin{aligned}&\mathbf{let}\ p = \mathbf{if}^\circ\ t\ \mathbf{then}\ u_0\ \mathbf{else}\ u_1\\ &\mathbf{in}\ e\\ &\equiv \mathbf{if}^\circ\ t\ \mathbf{then}\ (\mathbf{let}\ p = u_0\ \mathbf{in}\ e)\\ &\qquad\qquad \mathbf{else}\ (\mathbf{let}\ p = u_1\ \mathbf{in}\ e)\end{aligned}$$

Equations for **let**

$$\mathrm{Val}^{\mathrm{C}} ::= x \mid () \mid \mathit{false} \mid \mathit{true} \mid (\mathit{val}_1, \mathit{val}_2)$$

β

$$\textbf{let } p = \mathit{val} \textbf{ in } u \equiv u\,[\mathit{val} \,/\, p]$$

η

$$\textbf{let } x = t \textbf{ in } x \equiv t$$

Commuting conversion

$$\begin{aligned} &\textbf{let } p = t \textbf{ in let } q = u \textbf{ in } e \\ \equiv\ &\textbf{let } q = u \textbf{ in let } p = t \textbf{ in } e \end{aligned}$$

Quantum equations

(**if**$^\circ$)

$$\begin{aligned}&\textbf{if}^\circ\ (t_0 + t_1)\ \textbf{then}\ u_0\ \textbf{else}\ u_1\\ &\equiv (\textbf{if}^\circ\ t_0\ \textbf{then}\ u_0\ \textbf{else}\ u_1)\\ &+ (\textbf{if}^\circ\ t_1\ \textbf{then}\ u_0\ \textbf{else}\ u_1)\end{aligned}$$

$$\begin{aligned}&\textbf{if}^\circ\ (\lambda * t)\ \textbf{then}\ u_0\ \textbf{else}\ u_1\\ &\equiv \lambda * (\textbf{if}^\circ\ t\ \textbf{then}\ u_0\ \textbf{else}\ u_1)\end{aligned}$$

(superpositions)

$$\begin{aligned}t + u &\equiv u + t\\ t + \vec{0} &\equiv t\\ t + (u + v) &\equiv (t + u) + v\\ \lambda * (t + u) &\equiv \lambda * t + \lambda * u\\ \lambda * t + \kappa * t &\equiv (\lambda + \kappa) * t\\ 0 * t &\equiv \vec{0}\end{aligned}$$

QUICS questions

- Relation to the linear λ calculus by Paolo and Gilles ?
- Extend the theory to the full language (including measurements).
  related to Ross's question?
- Higher order?!

Motivation

- Explain quantum programming to (functional) programmers.
- Sell functional programming to people in quantum computing..
- Provide an intermediate language for the implementation of high level quantum languages (like QML).
- Framework to discover and implement patterns for quantum programming.

Haskell

- Pure functional programming language.
- Close to constructive Mathematics (terminating fragment).
- go further: Type Theory (Epigram).
- **Effects** (e.g. Input/Output, State, Concurrency, ...) are encapsulated in the IO monad.
- Proposal: Use *Functional specifications of IO* to reason about programs with IO.

Monads in Haskell

class *Monad m* **where**

$(\gg=) \in m\ a \rightarrow (a \rightarrow m\ b) \rightarrow m\ b$

$return \in a \rightarrow m\ a$

Equations:

$$\begin{aligned} return\ a \gg= f &= f\ a \\ c \gg= return &= c \\ (c \gg= f) \gg= g &= c \gg= \lambda a \rightarrow f\ a \gg= g \end{aligned}$$

Computations are represented by morphisms in the Kleisli category

$$a \rightarrow_{\text{Kleisli}} b = a \rightarrow m\ b$$

Haskell's IO monad

instance *Monad IO*

$$getChar \in IO\ Char$$
$$putChar \in Char \to IO\ ()$$

$$echo \in IO\ ()$$
$$echo = getChar \ggg\!= \lambda c \to putChar\ c \ggg\!= \lambda x \to echo$$

$$\begin{aligned} echo = \mathbf{do}\ & c \leftarrow getChar \\ & putChar\ c \\ & echo \end{aligned}$$

QIO

type *Qbit*

type *QIO a*
type *U*

instance *Monad QIO*

$$
\begin{aligned}
mkQbit &\in Bool \to QIO\ Qbit \\
applyU &\in U \to QIO\ () \\
meas &\in Qbit \to QIO\ Bool
\end{aligned}
$$

Reversible Ops

instance *Monoid U*

$unot \in Qbit \rightarrow U$
$uhad \in Qbit \rightarrow U$
$uphase \in Qbit \rightarrow \mathbb{R} \rightarrow U$
$swap \in Qbit \rightarrow Qbit \rightarrow U$
$cond \in Qbit \rightarrow (Bool \rightarrow U) \rightarrow U$

$cond\ x\ (\lambda b \rightarrow \mathbf{if}\ b\ \mathbf{then}\ unot\ x\ \mathbf{else}\ mempty)$

leads to a runtime error!

run or sim

- *run* embeds QIO into IO using a random number generator:

  $$run \in QIO\ a \to IO\ a$$

- or a real quantum computer...
- *sim* calculates the probability distribution of possible answers:

  $$sim \in QIO\ a \to Prob\ a$$

- where

  $$\mathbf{data}\ Prob\ a = Prob\ (\mathbf{Vec}\ \mathbb{R}\ a)$$

Example: a random bit

$$qran \in QIO\ Qbit$$

$$\begin{aligned} qran = \mathbf{do}\ & qb \leftarrow mkQbit\ \mathrm{True} \\ & applyU\ (uhad\ qb) \\ & return\ qb \end{aligned}$$

$$test\_qran \in QIO\ Bool$$

$$\begin{aligned} test\_qran = \mathbf{do}\ & qb \leftarrow qran \\ & meas\ qb \end{aligned}$$

$* Qio > run\ test\_qran$
False
$* Qio > run\ test\_qran$
True
$* Qio > sim\ test\_qran$
[(True, 0.5), (False, 0.5)]

The Bell state

$$
\begin{aligned}
&\mathit{share} \in \mathit{Qbit} \to \mathit{QIO}\ \mathit{Qbit}\\
&\mathit{share}\ \mathit{qa} = \mathbf{do}\ \mathit{qb} \leftarrow \mathit{mkQbit}\ \mathrm{False}\\
&\qquad\qquad\qquad \mathit{applyU}\ (\mathit{cond}\ \mathit{qa} \lambda a \to \mathbf{if}\ a\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{then}\ \mathit{unot}\ \mathit{qb}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \mathit{mempty})\\
&\qquad\qquad\qquad \mathit{return}\ \mathit{qb}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{bell} \in \mathit{QIO}\ (\mathit{Qbit}, \mathit{Qbit})\\
&\mathit{bell} = \mathbf{do}\ \mathit{qa} \leftarrow \mathit{qran}\\
&\qquad\qquad \mathit{qb} \leftarrow \mathit{share}\ \mathit{qa}\\
&\qquad\qquad \mathit{return}\ (\mathit{qa}, \mathit{qb})
\end{aligned}
$$

QUICS questions

- Measurement Calculus $\rightarrow$ QIO, or vice versa.
- Formal reasoning about QIO (factor through superoperators).

Hypotheses

- Interesting interactions between functional and quantum prgramming
- Design programming language as a vehicle to express high level patterns of quantum programming.
- Denotational semantics reflect our understanding of quantum programming.