Quantum Functional Programming Language & Its Denotational Semantics

Ichiro Hasuo
Dept. Computer Science
University of Tokyo

Naohiko Hoshino
Research Inst. for Math. Sci.
Kyoto University

Talk based on:
I. Hasuo & N. Hoshino,
Semantics of Higher-Order Quantum Computation via Geometry of Interaction,
to appear in Proc. Logic in Computer Science (LICS), June 2011.
Quantum Functional Programming Language
& Its Denotational Semantics
Quantum Functional Programming Language & Its Denotational Semantics
Quantum Functional Programming Language & Its Denotational Semantics

What?

Why?
Overview

• Why programming language?
• Why functional programming language?
• Why semantics?
• Why denotational semantics?
Overview

• Why programming language?
• Why functional programming language?
• Why semantics?
• Why denotational semantics?

Contribution
First denotational semantics for full-featured QFPL
Quantum Functional Programming Language & Its Denotational Semantics

Q1. Why programming language?
Formalisms

• We need one... for describing/studying quantum algorithms
Formalisms

- We need one... for describing/studying quantum algorithms

<table>
<thead>
<tr>
<th>Classical</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Boolean) circuit</td>
<td>Quantum circuit</td>
</tr>
<tr>
<td><img src="Null-Lobur" alt="Classical Circuit" /></td>
<td><img src="beachhandball.es" alt="Quantum Circuit" /></td>
</tr>
</tbody>
</table>

**Programming language**

```java
int i, j;
int factorial(int k)
{
    j=1;
    for (i=1; i<=k; i++)
        j=j*i;
    return j;
}
```
Formalisms

- We need one... for describing/studying quantum algorithms

<table>
<thead>
<tr>
<th>Classical</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Boolean) circuit</td>
<td>Quantum circuit</td>
</tr>
<tr>
<td><img src="Null-Lobur" alt="Classical circuit" /></td>
<td><img src="beachhandball.es" alt="Quantum circuit" /></td>
</tr>
<tr>
<td>Programming language</td>
<td>Quantum programming language</td>
</tr>
</tbody>
</table>
| ```c
int i,j;
int factorial(int k)
{
    j=1;
    for (i=1; i<=k; i++)
        j=j*i;
    return j;
}
``` | ```
telep = let (x,y) = EPR * in
let f = BellMeasure x in
let g = U y in (f,g).
``` |
Quantum Programming Languages

Imperative \cite{Mnarik}

```
void main() {
    qbit \psi_A, \psi_B;
    \psi_{EPR} aliasfor \{\psi_A, \psi_B\};
    channel[int] \epsilon \withends \{c_0, c_1\};
    \psi_{EPR} = createEPR();
    \epsilon = new channel[int]();
    fork bert(c_0, \psi_B);
    angela(c_1, \psi_A);
}
void angela(channelEnd[int] \epsilon, qbit \phi) {
    \phi = doSomething();
    r = measure (BellBasis, \phi, \epsilon);
    send (c_1, r);
}
```

Functional \cite{Selinger,Valiron}

```
telep = \begin{aligned}
    let \langle x, y \rangle &= EPR \cdot \in \\
    let f &= \text{BellMeasure} \cdot x \cdot \in \\
    let g &= U \cdot y \\
    in \langle f, g \rangle.
\end{aligned}
```

Figure 1: Teleportation implemented in LanQ
Quantum Programming Languages

Imperative (Minarik)

```
void main() {
    qbit psi_A, psi_B;
    psiEPR aliasfor [psi_A, psi_B];
    channel[int] e withends [c0, c1];
    psiEPR = createEPR();
    e = new channel[int]();
    fork bert(c0, psi_B);
    angela(c1, psi_A);
}
```

Functional (Selinger, Valiron)

```
qbit bert(channelEnd[int] c0, qbit stto) {
    int i;
    i = recv(c0);
    if (i == 0) {
        opB0(stto);
    } else if (i == 1) {
        opB1(stto);
    } else if (i == 2) {
        opB2(stto);
    } else {
        opB3(stto);
    }
    doSomethingElse(stto);
}
```

```
telep = let x, y = EPR * in
    let f = BellMeasure x in
    let g = U y
    in (f, g).
```

Figure 1: Teleportation implemented in LanQ

- “High-level” \(\rightarrow\) new algorithms?
Quantum Programming Languages

Imperative (Mlnarik)

- void main() {
  qbit $\psi_A$, $\psi_B$;
  $\psi_{EPR}$ alias for $[\psi_A, \psi_B]$;
  channel[int] $e$ with ends $[c_0, c_1]$;

- $\psi_{EPR} = \text{createEPR}()$;
- $e = \text{new channel}[\text{int}]()$;
- fork bert($c_0$, $\psi_B$);
- angela($c_1$, $\psi_A$);
}

- void angela(channelEnd[int] $c_1$, qbit $ata$) {
  int $r$;
  $\phi = \text{doSomething}()$;
  $r = \text{measure}(\text{BellBasis}, \phi, ata)$;
  send($c_1$, $t$);
}

Figure 1: Teleportation implemented in LanQ

Functional (Selinger, Valiron)

- telep = $\lambda (x, y) = \text{EPR} \ast \text{in}$
  
  let $f = \text{BellMeasure} x$ in
  
  let $g = U y$

  in $\langle f, g \rangle$.

- “High-level” $\rightarrow$ new algorithms?
- Well-developed techniques for correctness guarantee (verification)
  - Type system
  - Program model checking
  - etc.
Quantum Functional Programming Language
&
Its Denotational Semantics
Quantum Functional Programming Language & Its Denotational Semantics

Q2. Why *functional* programming language?
(Classical)
Functional Programming Languages

• Computation as *evaluation of mathematical functions*
• Avoids *(memory)* state or *mutable data*
• Scheme, Erlang, ML (SML, OCaml), Haskell, F#, ...

```c
int i, j;
int factorial(int k) {
    j = 1;
    for (i = 1; i <= k; i++)
        j = j * i;
    return j;
}
```

```
fun factorial x =
  if x = 0 then 1 else x * factorial (x-1)
```

Factorial in C

Factorial in ML

Hasuo (Tokyo), Hoshino (Kyoto)
(Classical)
Functional Programming Languages

• Higher-order computation

\[
twice \ f = \lambda x. f(fx)
\]
as

\[
\text{fun twice (f : int \to int) : int \to int = } \\
\text{fn (x : int) => f (f x)}
\]

• Modularity, code reusability
(Classical) Functional Programming Languages

- Higher-order computation

\[
\text{twice } f = \lambda x.f(fx)
\]

\[
\text{fun twice (f : int } \rightarrow \text{ int) : int } \rightarrow \text{ int } = \\
\text{ fn (x : int) } \Rightarrow \text{ f (f x)}
\]

- Modularity, code reusability

- Mathematically clean

- Programs as functions!
Quantum Functional Programming

- “Mathematical”
- $\rightarrow$ *Mathematical transfer* from classical to quantum
Quantum Functional Programming

- “Mathematical”
  - → Mathematical transfer from classical to quantum

- Uniform treatment of quantum data and classical data
Quantum Functional Programming

- “Mathematical”
- $\Rightarrow$ Mathematical transfer from classical to quantum
- Uniform treatment of quantum data and classical data

“quantum data, classical control”
Quantum Functional Programming

- "Mathematical"
- \( \Rightarrow \) Mathematical transfer from classical to quantum

- Uniform treatment of quantum data and classical data
- Nicely enforced by types

\[
0 : \text{int} \quad + : \text{int} \times \text{int} \rightarrow \text{int}
\]
Quantum Functional Programming

- “Mathematical”
  - $\rightarrow$ Mathematical transfer from classical to quantum

- Uniform treatment of quantum data and classical data

- Nicely enforced by types

```plaintext
0 : int      + : int * int -> int

new |0\> : qbit       tt : !bit
meas : qbit ~!bit
```

“quantum data, classical control”
Quantum Functional Programming

- “Mathematical”
  - → *Mathematical transfer* from classical to quantum

- Uniform treatment of *quantum data* and *classical data*

- Nicely enforced by *types*

```
0 : int      + : int * int -> int
new |0⟩ : qbit   tt : !bit
meas : qbit → !bit
```

“quantum data, classical control”
Quantum Functional Programming

- "Mathematical"
- \(\rightarrow\) Mathematical transfer from classical to quantum
- Uniform treatment of quantum data and classical data
- Nicely enforced by types

```
0 : int       + : int * int -> int
new |0\> : qbit  tt : !bit
meas : qbit --o !bit
```

- `!` : "duplicable"
- --o : "linear function" (input is used only once)
Our Language $q\lambda \ell$

- Based on [Selinger-Valiron, 2008]
- *Types:*
Our Language $q\lambda_\ell$

- Based on [Selinger-Valiron, 2008]
- **Types:**
  
  $A, B ::= n\text{-qbit} \mid \text{bit} \mid !A \mid A \to B \mid A \boxtimes B \mid \ldots$
Our Language $q\lambda$\textsubscript{\ell}

- Based on [Selinger and Valiron, 2008]
- **Types:**

$$A, B ::= \text{n-qbit} | \text{bit} | \neg A | A \rightarrow B | A \boxtimes B \ldots$$
Our Language $q\lambda_\ell$

- Based on [Selinger-Valiron, 2008]
- Types:
  
  \[
  A, B ::= \text{n-qubit} | \text{bit} | !A | A \rightarrow B | A \otimes B | \ldots
  \]
Our Language $q\lambda^e$

- Based on [Selinger, Valiron, 2008]
- **Types:**

  $A, B ::= n$-qubit $|$ bit $|$

  $!A | A \rightarrow B | A \otimes B | \ldots$

  $A$, duplicable

  $\lambda$-calculus (prototype FPL)

  n-qubit state

  (classical) bit

  Hasuo (Tokyo), Hoshino (Kyoto)
Our Language $q\lambda^e$  

- Based on [Selinger-Valiron, 2008]

**Types:**

\[ A, B ::= n\text{-qbit} \mid \text{bit} \mid !A \mid A \to B \mid A \otimes B \mid \ldots \]

- $n$-qubit state
- (classical) bit
- $A$, duplicable
- Linear function

\text{$\lambda$-calculus (prototype FPL)}
Our Language $q\lambda^\ell$

• Based on [Selinger-Valiron, 2008]

• Types:

$A, B ::= n$-qubit $| \text{bit} |$

$!A | A \rightarrow B | A \otimes B | \ldots$

$A$, duplicable

linear function

pair of $A \& B$

n-qubit state

(\text{classical}) \text{ bit}$

\begin{itemize}
  \item $\lambda$-calculus (prototype FPL)
\end{itemize}
Our Language $q\lambda_l$

- Based on [Selinger-Valiron, 2008]

- **Types:**
  - $A, B ::= n$-qubit | bit | $\textit{!} A | A \rightarrow B | A \otimes B | \ldots$

- **Programs or terms:**
  - $M, N ::= x \in \text{Var} | \lambda x^A.M | MN | \langle M, N \rangle | \text{let } \langle x^A, y^B \rangle = M \text{ in } N$
  - $\text{letrec } f^A x = M \text{ in } N | \text{new } |0\rangle | \text{meas}_i^{n+1} | U | \text{cmp}_{m,n}$

$\lambda$-calculus (prototype FPL)

Hasuo (Tokyo), Hoshino (Kyoto)
Our Language $q\lambda^\ell$

- Based on [Selinger-Valiron, 2008]

- **Types:**
  
  $A, B ::= n\text{-qbit} \mid \text{bit} \mid !A \mid A \rightarrow B \mid A \otimes B \mid \ldots$

- **Programs or terms:**
  
  $M, N ::= $
  
  $x \in \text{Var} \mid \lambda x^A.M \mid MN \mid \langle M, N \rangle \mid \text{let} \langle x^A, y^B \rangle = M \text{ in } N$
  
  letrec $f^A x = M \text{ in } N \mid$
  
  new $|0\rangle \mid \text{meas}_{i}^{n+1} \mid U \mid \text{cmp}_{m,n}$

\(\lambda\)-calculus (prototype FPL)

Hasuo (Tokyo), Hoshino (Kyoto)

Monday, November 7, 2011
Our Language $q\lambda_l$

- Based on [Selinger-Valiron, 2008]
- Types:
  \[ A, B ::= n\text{-qubit} \mid \text{bit} \mid \]
  \[ ! A \mid A \rightarrow B \mid A \otimes B \mid \ldots \]
- Programs or terms:
  \[ M, N ::= x \in \text{Var} \mid \lambda x^A.M \mid MN \mid \]
  \[ \langle M, N \rangle \mid \text{let} \langle x^A, y^B \rangle = M \text{ in } N \]
  \[ \text{letrec } f^A x = M \text{ in } N \mid \]
  \[ \text{new}|0\rangle \mid \text{meas}_{i}^{n+1} \mid U \mid \text{cmp}_{m,n} \]
Our Language $q\lambda^\ell$

- Based on [Selinger-Valiron, 2008]
- **Types:**
  \[ A, B ::= \text{n-qbit} | \text{bit} | !A | A \to B | A \otimes B | \ldots \]
- **Programs or terms:**
  \[ M, N ::= \]
  \[
  x \in \text{Var} | \lambda x^A.M | MN |
  \langle M, N \rangle | \text{let } \langle x^A, y^B \rangle = M \text{ in } N
  \text{letrec } f^A x = M \text{ in } N | \\
  \text{new } |0\rangle | \text{meas}_{i}^{n+1} | U | \text{cmp}_{m,n}
  \]

\[\lambda\text{-calculus (prototype FPL)}\]

Hasuo (Tokyo), Hoshino (Kyoto)
Our Language $q\lambda_{\ell}$

- Based on [Selinger-Valiron, 2008]
- **Types:**
  \[ A, B ::= \text{n-qubit state} | \text{bit} | !A | A \rightarrow B | A \otimes B | \ldots \]

- **Programs or terms:**
  \[ M, N ::= x \in \text{Var} | \lambda x^A.M | MN | \langle M, N \rangle | \text{let} \langle x^A, y^B \rangle = M \text{ in } N \]
  \[ \text{letrec } f^A x = M \text{ in } N | \text{new } |0\rangle | \text{meas}_{i}^{n+1} | U | \text{cmp}_{m,n} \]
Our Language $q\lambda^\ell$

• **Typing rules:** N.B. Only some are shown. Very much simplified

\[
\frac{\! \Delta, x : A \vdash x : A}{\! \Delta, x : A \vdash x : A} \quad (Ax.1)
\]

\[
\frac{\! \Delta \vdash \text{new} \mid 0 : \text{qbit}}{\! \Delta \vdash \text{new} \mid 0 : \text{qbit}} \quad (Ax.2) \quad \frac{\! \Delta \vdash \text{meas} : ! (\text{qbit} \to \circ \text{bit})}{\! \Delta \vdash \text{meas} : ! (\text{qbit} \to \circ \text{bit})} \quad (Ax.2)
\]

\[
\frac{x : A, \Delta \vdash M : B}{\Delta \vdash \lambda x^A. M : A \to \circ B} \quad (\circ.I_1)
\]

\[
\frac{\! \Delta, \Gamma_1 \vdash M : A \to \circ B \quad \! \Delta, \Gamma_2 \vdash N : A}{\! \Delta, \Gamma_1, \Gamma_2 \vdash MN : B} \quad (\circ.E), (\dag)
\]

\[
\frac{\! \Delta, \Gamma_1 \vdash M_1 : A_1 \quad \! \Delta, \Gamma_2 \vdash M_2 : A_2}{\! \Delta, \Gamma_1, \Gamma_2 \vdash \langle M_1, M_2 \rangle : A_1 \, \boxtimes \, A_2} \quad (\boxtimes.I), (\dag)
\]
Our Language $q\lambda^\ell$

• **Typing rules:** N.B. Only some are shown. Very much simplified

\[
\frac{!\Delta, x : A \vdash x : A}{(Ax.1)}
\]

\[
\frac{!\Delta \vdash \text{new} |0\rangle : \text{qbit}}{(Ax.2)}
\]

\[
\frac{!\Delta \vdash \text{meas} : !(\text{qbit} \to \text{bit})}{(Ax.2)}
\]

\[
\frac{x : A, \Delta \vdash M : B}{\Delta \vdash \lambda x^A.M : A \to B} \quad (-\circ.I_1)
\]

\[
\frac{!\Delta, \Gamma_1 \vdash M : A \to B}{!\Delta, \Gamma_1, \Gamma_2 \vdash MN : B} \quad \frac{!\Delta, \Gamma_2 \vdash N : A}{(-\circ.E), (\dagger)}
\]

\[
\frac{!\Delta, \Gamma_1 \vdash M_1 : A_1}{!\Delta, \Gamma_1, \Gamma_2 \vdash \langle M_1, M_2 \rangle : A_1 \boxtimes A_2 \quad (\bowtie.I), (\dagger)}
\]
Type Discipline

- Typable $\rightarrow$ “safe”
- Guarantees minimal “correctness”

\[
\vdash f^{A \rightarrow B} x^A : B \quad \forall f^{A \rightarrow B} y^A \rightarrow A
\]

\[
\vdash \text{meas}(\sqrt{\text{qbit}}) : \text{bit} \quad \forall \langle \text{meas}(\sqrt{\text{qbit}}), \text{meas}(H \sqrt{\text{qbit}}) \rangle
\]
Type Discipline

- Typable $\rightarrow$ “safe”

- Guarantees minimal “correctness”

$$ \vdash f : A \rightarrow B \ x : A : B \quad \forall \ f : A \rightarrow B \ y : A : A $$

$$ \vdash \text{meas}(x^{\text{qbit}}) : \text{bit} \quad \forall \langle \text{meas}(x^{\text{qbit}}), \text{meas}(H x^{\text{qbit}}) \rangle $$

**Faulty program**

```
fun isValue t =
  case t of
    Num _ => 1
  | _ => false
```

**Type error**

```
ex.sml:22.3-24.15 Error: types of rules don't agree
[literal]
  earlier rule(s): term -> int
  this rule: term -> bool
  in rule:
    _ => false
```

Hasuo (Tokyo), Hoshino (Kyoto)
Examples

- Quantum teleportation

\[
\begin{align*}
EPR &= \lambda x.\text{CNOT}(H(\text{new } 0), \text{new } 0) \\
\text{BellMeasure} &= \\
&\lambda q_2.\lambda q_1. (\text{let } x, y \rightarrow \text{CNOT}(q_1, q_2) \text{ in } \langle \text{meas}(Hx), \text{meas } y \rangle) \\
U &= \lambda q.\lambda (x, y). \text{if } x \text{ then } (\text{if } y \text{ then } U_{11}q \text{ else } U_{10}q) \\
&\quad \text{else } (\text{if } y \text{ then } U_{01}q \text{ else } U_{00}q). \\
\text{telep} &= \text{let } (x, y) = \text{EPR} \ast \text{in} \\
&\quad \text{let } f = \text{BellMeasure } x \text{ in} \\
&\quad \text{let } g = U \ y \\
&\quad \text{in } \langle f, g \rangle.
\end{align*}
\]

- (Fair) cointoss, repeated Hadamard

\[
\begin{align*}
c &= \lambda \ast. \text{meas}(H(\text{new } 0)) \\
M &= \text{let } \text{rec } f x = (\text{if } (c \ast) \text{ then } H(f x) \text{ else } x) \text{ in } f \ p
\end{align*}
\]

Flip coin:
- head ➔ Hadamard and flip again
- tail ➔ done

Hasuo (Tokyo), Hoshino (Kyoto)
Quantum Functional Programming Language
&
Its Denotational Semantics
Quantum Functional Programming Language & Its Denotational Semantics

Q3. Why semantics?
Semantics
Semantics

• “Meaning” of a program
Semantics

• “Meaning” of a program

• For reasoning about programs
Semantics

• “Meaning” of a program

• For *reasoning about* programs

• $M \equiv N$: “$M$ and $N$ have the same meaning, i.e. computational content”
Semantics

• “Meaning” of a program

• For reasoning about programs

  • $M \equiv N$: “$M$ and $N$ have the same meaning, i.e. computational content”

    \[
    \lambda x. (x - x) \equiv \lambda x. 0
    \]
Semantics

• “Meaning” of a program

• For reasoning about programs

• $M \equiv N$: “$M$ and $N$ have the same meaning, i.e. computational content”

$$\lambda x. (x - x) \triangleq \lambda x. 0$$

(stupid) sort $\equiv$ quick sort
Semantics

• For functional languages:
  
  • **Operational**: how the program is transformed/evaluated/reduced
    
    
    \[(\lambda x. 1 + x)3 \rightarrow 1 + 3 \rightarrow 4\]
    
  • **Denotational**: “meaning” as a mathematical function
    
    \[\[\lambda x. 1 + x\] = (\text{function } \mathbb{N} \rightarrow \mathbb{N}, \ n \mapsto 1 + n)\]
# Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\lambda x. 1 + x)^3 \rightarrow 1 + 3 \rightarrow 4$</td>
<td>$[\lambda x. 1 + x] = \text{(function } \mathbb{N} \rightarrow \mathbb{N}, ; n \mapsto 1 + n)$</td>
</tr>
<tr>
<td>reduction-based</td>
<td>“mathematical”</td>
</tr>
<tr>
<td>dynamic</td>
<td>static</td>
</tr>
</tbody>
</table>
## Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda x. 1 + x)3 \rightarrow 1 + 3 \rightarrow 4)</td>
<td>([\lambda x. 1 + x] = (\text{function } \mathbb{N} \rightarrow \mathbb{N}, \ n \mapsto 1 + n))</td>
</tr>
<tr>
<td>reduction-based \textit{dynamic}</td>
<td>“mathematical” \textit{static}</td>
</tr>
<tr>
<td>(akin to) machine implementation</td>
<td>comes with \textit{mathematical reasoning principles} (fixed pt. induction, well-fdd induction, etc.)</td>
</tr>
</tbody>
</table>
## Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>(λx. 1 + x)³ → 1 + 3 → 4</td>
<td>[λx. 1 + x] = (function N → N, n ↦ 1 + n)</td>
</tr>
<tr>
<td>reduction-based dynamic</td>
<td>“mathematical static”</td>
</tr>
<tr>
<td>(akin to) machine implementation</td>
<td>comes with mathematical reasoning principles (fixed pt. induction, well-fdd induction, etc.)</td>
</tr>
</tbody>
</table>
## Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>(λx. 1 + x)³ → 1 + 3 → 4</td>
<td>[λx. 1 + x] = (function ( \mathbb{N} \rightarrow \mathbb{N} ), ( n \mapsto 1 + n ))</td>
</tr>
<tr>
<td>reduction-based</td>
<td>“mathematical” static</td>
</tr>
<tr>
<td><em>dynamic</em></td>
<td>comes with <em>mathematical reasoning principles</em> (fixed pt. induction, well-fdd induction, etc.)</td>
</tr>
<tr>
<td>(akin to) machine implementation</td>
<td>powerful esp. for recursion</td>
</tr>
</tbody>
</table>

Goal:

\[ M \cong_{\text{opr.}} N \]
## Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\lambda x. 1 + x)3 \rightarrow 1 + 3 \rightarrow 4)</td>
<td>([\lambda x. 1 + x] = \text{(function N \rightarrow N, } n \mapsto 1 + n))</td>
</tr>
</tbody>
</table>

- **Operational**: reduction-based, **dynamic**
- **Denotational**: "mathematical", **static**

(akin to) machine implementation

---

**Goal:**

\[ M \approx_{\text{opr.}} N \]

---

*Hasuo (Tokyo), Hoshino (Kyoto)*

Monday, November 7, 2011
# Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>(λx. 1 + x)³ → 1 + 3 → 4</td>
<td>[λx. 1 + x] = (function N → N, n ↦ 1 + n)</td>
</tr>
<tr>
<td>reduction-based</td>
<td>“mathematical”</td>
</tr>
<tr>
<td>dynamic</td>
<td>static</td>
</tr>
<tr>
<td>(akin to) machine implementation</td>
<td>comes with mathematical reasoning principles</td>
</tr>
<tr>
<td></td>
<td>(fixed pt. induction, well-fdd induction, etc.)</td>
</tr>
</tbody>
</table>

**Goal:**

\[ M \cong_{\text{opr.}} N \]

---

Hasuo (Tokyo), Hoshino (Kyoto)

Monday, November 7, 2011
# Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>(λx. 1 + x)3 → 1 + 3 → 4</td>
<td>[λx. 1 + x] = (function N → N, n ↦ 1 + n)</td>
</tr>
<tr>
<td>reduction-based</td>
<td>“mathematical”</td>
</tr>
<tr>
<td>dynamic</td>
<td><em>static</em></td>
</tr>
<tr>
<td>(akin to) machine</td>
<td>comes with <em>mathematical reasoning principles</em></td>
</tr>
<tr>
<td>implementation</td>
<td>(fixed pt. induction, well-fdd induction, etc.)</td>
</tr>
</tbody>
</table>

Goal:

\[ M \cong_{\text{opr.}} N \]

hard to show directly

Hasuo (Tokyo), Hoshino (Kyoto)
## Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>(akin to) machine implementation</td>
<td>“mathematical reasoning principles”</td>
</tr>
<tr>
<td>reduction-based dynamic</td>
<td>comes with mathematical reasoning principles</td>
</tr>
<tr>
<td>(λx. 1 + x)(\mathbb{N}) → 1 + 3 → 4</td>
<td>(\lfloor \lambda x. 1 + x \rfloor = \text{(function } \mathbb{N} \to \mathbb{N}, \ n \mapsto 1 + n))</td>
</tr>
</tbody>
</table>

**Goal:**

\[
\begin{align*}
M & \cong_{\text{opr.}} N \\
\iff & \quad [M] = [N]
\end{align*}
\]

- hard to show directly

---

Hasuo (Tokyo), Hoshino (Kyoto)
Operational vs. Denotational

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\lambda x. 1 + x) 3 \rightarrow 1 + 3 \rightarrow 4$</td>
<td>$[\lambda x. 1 + x] = \text{(function } \mathbb{N} \rightarrow \mathbb{N}, \ n \mapsto 1 + n)$</td>
</tr>
<tr>
<td>reduction-based, dynamic</td>
<td>“mathematical”, static</td>
</tr>
<tr>
<td>(akin to) machine implementation</td>
<td>comes with mathematical reasoning principles</td>
</tr>
<tr>
<td></td>
<td>$(\text{fixed pt. induction, well-fdd induction, etc.})$</td>
</tr>
</tbody>
</table>

Goal:

$M \cong_{\text{opr.}} N$ \iff \quad [M] = [N] \iff \quad M \cong_{\text{opr.}} N$

Adequate denotational semantics:

$[M] = [N] \iff M \cong_{\text{opr.}} N$

Monday, November 7, 2011
Quantum Functional Programming Language & Its Denotational Semantics
Quantum Functional Programming Language & Its Denotational Semantics

Q4. Why no denotational semantics before?
Challenges

\[
[H] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} : \mathbb{C}^2 \rightarrow \mathbb{C}^2 , \text{ isn’t it?}
\]
Challenges

\[ [H] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} : \mathbb{C}^2 \rightarrow \mathbb{C}^2, \text{ isn’t it?} \]

- “Quantum data, classical control”
- \( \rightarrow \) not clear how to accommodate duplicable data in Hilbert spaces
Technical Contributions
Technical Contributions

- Quantum functional programming language
- Based on [Selinger-Valiron]
- w/ recursion, classical data (by !)
Technical Contributions

- Quantum functional programming language
  - Based on [Selinger-Valiron]
  - w/ recursion, classical data (by !)

- Its denotational semantics
  - First one for fully-featured QFPL
Full-fledged Semantical Technologies

- Monad $B$ for branching
  \[ \Downarrow \text{Take the Kleisli category} \]
  - Traced monoidal category
    \[ \Downarrow \text{Int-construction, [9]} \]
    - Compact closed category
      \[ \Downarrow \text{Find a reflexive object} \]
      - Linear combinatorial algebra $A$
        \[ \Downarrow \text{Take } \text{PER}_A, \text{ the category of partial equivalence relations} \]
      - Linear category that models computation

Categorical GoI [7]

Hasuo (Tokyo), Hoshino (Kyoto)
Full-fledged Semantical Technologies

Monad $B$ for branching

Take the Kleisli category

Traced monoidal category

Int-construction, [9]

Compact closed category

Find a reflexive object

Linear combinatorial algebra $A$

Take $\text{PER}_A$, the category of partial equivalence relations

Linear category that models computation

Categorical GoI [7]
Geometry of Interaction

- Originally by J.-Y. Girard, 1989:
  - Computation as *player of a game*
- cf. Game semantics (Abramsky et al., Hyland-Ong)
- We use *categorical* formulation: Abramsky, Haghverdi and Scott, 2002
Geometry of Interaction

- Originally by J.-Y. Girard, 1989:
  - Computation as *player of a game*
  - cf. Game semantics (Abramsky et al., Hyland-Ong)
  - We use *categorical* formulation: Abramsky, Haghverdi and Scott, 2002

- *Axiomatization of what is “classical control”*
(Particle-Style)
Geometry of Interaction

$$[M] = \begin{array}{cccc}
\downarrow & \downarrow & \downarrow & \downarrow \\
1 & 2 & 3 & 4 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
\end{array} \ldots \text{(countably many)}$$
(Particle-Style) Geometry of Interaction

\[
\begin{bmatrix} M \end{bmatrix} = \begin{array}{c}
\downarrow \downarrow \downarrow \downarrow \\
1 \quad 2 \quad 3 \quad 4 \\
\end{array} \quad \ldots \quad (\text{countably many})
\]

Hasuo (Tokyo), Hoshino (Kyoto)
(Particle-Style)
Geometry of Interaction

\[
\begin{bmatrix} M \end{bmatrix} = M
\]

Hasuo (Tokyo), Hoshino (Kyoto)

Monday, November 7, 2011
(Particle-Style)
Geometry of Interaction

"token" (chocolate)

\[
[M] =
\]

Hasuo (Tokyo), Hoshino (Kyoto)

Monday, November 7, 2011
(Particle-Style)

Geometry of Interaction

\[
[M] = M
\]

1 2 3 4 ...

(countably many)
(Particle-Style)
Geometry of Interaction

$$[M] = M$$

Hasuo (Tokyo), Hoshino (Kyoto)
(Particle-Style) Geometry of Interaction

\[
\begin{array}{cccc}
\downarrow & \downarrow & \downarrow & \downarrow \\
1 & 2 & 3 & 4 \\
\end{array}
\]

\[ [M] = \]

\[ M \]

... (countably many)

Hasuo (Tokyo), Hoshino (Kyoto)

Monday, November 7, 2011
\[ [M \ N] = \begin{array}{c}
\ldots \\
M \\
\ldots \\
\end{array} \\
\begin{array}{c}
\ldots \\
N \\
\ldots \\
\end{array} \]
\[ \begin{bmatrix} M & N \end{bmatrix} = \begin{array}{c} \cdots \cdots \\ \hline M \\ \hline \cdots \\ \hline \end{array} = \begin{array}{c} \cdots \\ \hline N \\ \hline \cdots \\ \hline \end{array} \]
\[ [MN] = M \]
\[ [M N] = M \cdot N \]
\[ MN \] = 

\[ M \] 

\[ N \]
\[ [MN] \]

\[
\begin{align*}
M &= \lambda x. x + 1 \\
N &= 2 \\
M &= \lambda x. 1 \\
N &= 2 \\
M &= \lambda f. f 1 \\
N &= \lambda x. (x + 1)
\end{align*}
\]
\[ [MN] = \]

\[ M = \lambda x. x + 1 \quad N = 2 \]
\[ M = \lambda x. 1 \quad N = 2 \]
\[ M = \lambda f. f1 \quad N = \lambda x. (x + 1) \]
\[ [MN] = \]

\[ M = \lambda x. x + 1 \quad N = 2 \]
\[ M = \lambda x. 1 \quad N = 2 \]
\[ M = \lambda f. f1 \quad N = \lambda x. (x + 1) \]

Monday, November 7, 2011
\[ [MN] = \]

\[ M = \lambda x. x + 1 \quad N = 2 \]
\[ M = \lambda x. 1 \quad N = 2 \]
\[ M = \lambda f. f1 \quad N = \lambda x. (x + 1) \]
\[ [MN] = \] 

... 

\[MN\] 

\[M = \lambda x. x + 1 \quad N = 2\] 
\[M = \lambda x. 1 \quad N = 2\] 
\[M = \lambda f. f1 \quad N = \lambda x. (x + 1)\]
\[ MN \] = ... 

\[ M = \lambda x. x + 1 \quad N = 2 \]
\[ M = \lambda x. 1 \quad N = 2 \]
\[ M = \lambda f. f1 \quad N = \lambda x. (x + 1) \]
\[ MN \]

... 

\[
M = \lambda x. x + 1 \\
N = 2 \\
\]

\[
M = \lambda f. f1 \\
N = \lambda x. (x + 1) \\
\]

Monday, November 7, 2011
\[ [M N] = \]

\[ M = \lambda x. x + 1 \quad N = 2 \]

\[ M = \lambda x. 1 \quad N = 2 \]

\[ M = \lambda f. f1 \quad N = \lambda x. (x + 1) \]
\[ MN \]

\[ = \]

\[ M = \lambda x. x + 1 \quad N = 2 \]

\[ M = \lambda x. 1 \quad N = 2 \]

\[ M = \lambda f. f1 \quad N = \lambda x. (x + 1) \]

Monday, November 7, 2011
\[ MN \]

\[ \begin{align*}
M &= \lambda x. x + 1 \\
N &= 2
\end{align*} \]

\[ \begin{align*}
M &= \lambda x. 1 \\
N &= 2
\end{align*} \]

\[ \begin{align*}
M &= \lambda f. f1 \\
N &= \lambda x. (x + 1)
\end{align*} \]
\[ [MN] = \]

\[ M = \lambda x. x + 1 \quad N = 2 \]
\[ M = \lambda x. 1 \quad N = 2 \]
\[ M = \lambda f. f1 \quad N = \lambda x. (x + 1) \]
\[ [MN] = \]

\[ \begin{align*}
M &= \lambda x. x + 1 \\
N &= 2 \\
M &= \lambda x. 1 \\
N &= 2 \\
M &= \lambda f. f1 \\
N &= \lambda x. (x + 1)
\end{align*} \]
\[
[MN] = M = \lambda x. x + 1 \quad N = 2 \\
M = \lambda x. 1 \quad N = 2 \\
\rightarrow M = \lambda f. f1 \quad N = \lambda x. (x + 1)
\]
\[ MN \]

\[ = \]

\[ M = \lambda x. x + 1 \quad N = 2 \]
\[ M = \lambda x. 1 \quad N = 2 \]
\[ M = \lambda f. f1 \quad N = \lambda x. (x + 1) \]
Quantum
Geometry of Interaction

\[
[M] = M
\]

1 2 3 4 ...

(countably many)

Hasuo (Tokyo), Hoshino (Kyoto)

Monday, November 7, 2011
Quantum

Geometry of Interaction

Not just a token/particle, but quantum state!

\[ [M] = 

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
\end{array}
\]

\( \cdots \) (countably many)

\[ M \]

\[ \cdots \]
Quantum Geometry of Interaction

\[
[M] = M
\]

Not just a token/particle, but \textit{quantum state}!
Quantum Geometry of Interaction

$[M] = M$

“Quantum Data”

Not just a token/particle, but quantum state!
Quantum Geometry of Interaction

\[ [M] = \]

“Quantum Data”

“Classical Control”

Not just a token/particle, but quantum state!

1 2 3 4 ...
(countably many)

Hasuo (Tokyo), Hoshino (Kyoto)

Monday, November 7, 2011
Quantum Functional Programming Language
&
Its Denotational Semantics
Quantum Functional Programming Language & Its Denotational Semantics

Q5. Why quantum computation?
Ans. You know why!
Conclusions

• Structured programming & mathematical semantics

• Quantum data, classical control

• *Geometry of interaction* as the essence of classical control
Conclusions

• Structured programming & mathematical semantics

• Quantum data, classical control

• Geometry of interaction as the essence of classical control

Thank you for your attention!
Ichiro Hasuo (Dept. CS, U Tokyo)
http://www-mmm.is.s.u-tokyo.ac.jp/~ichiro/

Naohiko Hoshino (RIMS, Kyoto U)
http://www.kurims.kyoto-u.ac.jp/~naophiko/
Quantum Programming Languages

Functional (Selinger, Valiron)

```
telep = let (x, y) = EPR * in
  let f = BellMeasure x in
  let g = U y
  in (f, g).
```
Quantum Programming Languages

Imperative (Mlnarik)

```c
void main() {
    qbit $\psi_A$, $\psi_B$;
    $\psi_{EPR}$ aliasfor [$\psi_A$, $\psi_B$];
    channel[int] $e$ withends [c_0, c_1];
    $\psi_{EPR}$ = createEPR();
    $e$ = new channel[int]();
    fork bert(c_0, $\psi_B$);
    angela(c_1, $\psi_A$);
}

void angela(channelEnd[int] $c_1$, qbit $ata$) {
    int $t$;
    qbit $\phi$;
    $\phi$ = doSomething();
    $r$ = measure (BellBasis, $\phi$, $ata$);
    send ($c_1$, $t$);
}
```

Functional (Selinger, Valiron)

```c
qbit bert(channelEnd[int] $c_0$, qbit $stto$) {
    int $i$;
    $i$ = recv ($c_0$);
    if ($i$ = 0) {
        $op_{B_0}$($stto$);
    } else if ($i$ = 1) {
        $op_{B_1}$($stto$);
    } else if ($i$ = 2) {
        $op_{B_2}$($stto$);
    } else {
        $op_{B_3}$($stto$);
    }
    doSomethingElse($stto$);
}
```

telep = let $(x, y) = EPR \ast in$
        let $f = BellMeasure x in$
        let $g = U y$
        in $(f, g)$.

Figure 1: Teleportation implemented in LanQ

Hasuo (Tokyo), Hoshino (Kyoto)
Quantum Programming Languages

Imperative (Mlnarik)

```
void main() {
    qbit ψA, ψB;
    ψEPR aliasfor [ψA, ψB];
    channel[int] e withends [c0, c1];
    ψEPR = createEPR();
    e = new channel[int]();
    fork bert(c0, ψB);
    angela(c1, ψA);
}
```

```
void angela(channel[int] c1, qbit ata) {
    int r;
    qbit ϕ;
    ϕ = doSomething();
    r = measure (BellBasis, ϕ, ata);
    send (c1, r);
}
```

Functional (Selinger; Valiron)

```
qbit bert(channelEnd[int] c0, qbit stto) {
    int i;
    i = recv (c0);
    if (i == 0) {
        opB0(stto);
    } else if (i == 1) {
        opB1(stto);
    } else if (i == 2) {
        opB2(stto);
    } else {
        opB3(stto);
    }
    doSomethingElse(stto);
}
```

```
telep = let \langle x, y \rangle = EPR * in
let f = BellMeasure x in
let g = U y
    in \langle f, g \rangle.
```

Figure 1: Teleportation implemented in LanQ

- “High-level” → new algorithms?
Quantum Programming Languages

- “High-level” → new algorithms?
- (Sometimes) good handling of quantum vs. classical data
  - No-Cloning vs. Duplicable
Quantum Programming Languages

- "High-level" → new algorithms?
- (Sometimes) good handling of quantum vs. classical data
- No-Cloning vs. Duplicable
- Model quantum communication protocols

Imperative (Mlarik)

```c
void main() {
  qbit ψ_A, ψ_B;
  ψ_EPR aliasfor [ψ_A, ψ_B];
  channel[| ] withends [c0, c1]
  ψ_EPR = createEPR();
  e = new channel[| ];
  fork bert(c0, ψ_B);
  angela(c1, ψ_A);
}

void angela(channelEnd[int] c1, qbit at) {
  int r;
  qbit φ;
  φ = doSomething();
  r = measure (BellBasis, φ, at);
  send (c1, r);
}
```

Functional (Selinger, Valiron)

```
telep = let <x, y> = EPR * in
   let f = BellMeasure x in
   let g = U y
    in <f, g>.
```

Figure 1: Teleportation implemented in LanQ

Hasuo (Tokyo), Hoshino (Kyoto)
Quantum Functional Programming Language
&
Its Denotational Semantics
Quantum Functional Programming Language & Its Denotational Semantics

Q4. Why denotational semantics?