The Semantics of Effects: Centrality, Quantum Control and Reversible Recursion

Louis LEMONNIER

Supervised by: Pablo ARRIGHI, Benoît VALIRON, Vladimir ZAMDZHIJEV

PhD Defence. 19th June 2024
\( \lambda \)-calculus:

- Form a function: \( f = \lambda x. x + 2 \equiv x \mapsto x + 2 \)
- Apply the function: \( f (3) \), or \( f 3 \).
**Introduction: Languages and Types**

\(\lambda\)-calculus:

- Form a function: \( f = \lambda x. x + 2 \equiv x \mapsto x + 2 \)
- Apply the function: \( f(3) \), or \( f\ 3 \).

A type \( A \) is a specification of a term \( t \), we write \( t : A \).

For example:

\[
\text{True: Bool} \quad 2: \text{Nat} \quad \lambda x. x + 2: \text{Nat \rightarrow Nat}
\]
\[\lambda\text{-calculus:}\]

- Form a function: \( f = \lambda x. x + 2 \equiv x \mapsto x + 2 \)
- Apply the function: \( f(3) \), or \( f\,3 \).

A type \( A \) is a specification of a term \( t \), we write \( t : A \).

For example:

- \( \text{True: Bool} \)
- \( 2 : \text{Nat} \)
- \( \lambda x. x + 2 : \text{Nat} \rightarrow \text{Nat} \)

We also have variables:

- \( \text{if } x \text{ then } y \text{ else } y + 2 \)

Example of judgement:

- \( x : \text{Bool}, y : \text{Nat} \vdash \text{if } x \text{ then } y \text{ else } y + 2 : \text{Nat} \)
Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let * = t in t'</td>
</tr>
</tbody>
</table>
## Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let * = t in t'</td>
</tr>
<tr>
<td>A → B</td>
<td>λx.t</td>
<td>tt'</td>
</tr>
</tbody>
</table>

General inductive types:

| X : A | Nat = X | List (A) = X : I | A List (A) = X : I |

```
## Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let * = t in t'</td>
</tr>
<tr>
<td>A → B</td>
<td>( \lambda x. t )</td>
<td>tt'</td>
</tr>
<tr>
<td>A ⊗ B</td>
<td>( \langle t_1, t_2 \rangle )</td>
<td>let ( \langle x, y \rangle = t ) in t'</td>
</tr>
</tbody>
</table>
### Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let * = t in t'</td>
</tr>
<tr>
<td>A → B</td>
<td>λx.t</td>
<td>tt'</td>
</tr>
<tr>
<td>A ⊗ B</td>
<td>⟨t₁, t₂⟩</td>
<td>let ⟨x, y⟩ = t in t'</td>
</tr>
<tr>
<td>Nat</td>
<td>zero, S t</td>
<td>match t with {zero \mapsto t₁</td>
</tr>
</tbody>
</table>

General inductive types:

- Nat = I
- List(A) = I(A, List(A))
## Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let * = t in t'</td>
</tr>
<tr>
<td>A → B</td>
<td>λx.t</td>
<td>tt'</td>
</tr>
<tr>
<td>A ⊗ B</td>
<td>⟨t₁, t₂⟩</td>
<td>let ⟨x, y⟩ = t in t'</td>
</tr>
<tr>
<td>Nat</td>
<td>zero, S t</td>
<td>match t with {zero ↦ t₁</td>
</tr>
<tr>
<td>List(A)</td>
<td>Empty, Cons ⟨h, t⟩</td>
<td>match t with {Empty ↦ t₁</td>
</tr>
</tbody>
</table>
## Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let * = t in t'</td>
</tr>
<tr>
<td>A → B</td>
<td>λx.t</td>
<td>tt'</td>
</tr>
<tr>
<td>A ⊗ B</td>
<td>⟨t₁, t₂⟩</td>
<td>let ⟨x, y⟩ = t in t'</td>
</tr>
<tr>
<td>Nat</td>
<td>zero, S t</td>
<td>match t with {zero ↦ t₁</td>
</tr>
<tr>
<td>List(A)</td>
<td>Empty, Cons ⟨h, t⟩</td>
<td>match t with {Empty ↦ t₁</td>
</tr>
<tr>
<td>A ⊕ B</td>
<td>left t, right t</td>
<td>match t with {left ↦ t₁</td>
</tr>
</tbody>
</table>
### Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let * = t in t'</td>
</tr>
<tr>
<td>A → B</td>
<td>λx.t</td>
<td>tt'</td>
</tr>
<tr>
<td>A ⊗ B</td>
<td>⟨t₁, t₂⟩</td>
<td>let ⟨x, y⟩ = t in t'</td>
</tr>
<tr>
<td>Nat</td>
<td>zero, S t</td>
<td>match t with {zero ↦ t₁</td>
</tr>
<tr>
<td>List(A)</td>
<td>Empty, Cons ⟨h, t⟩</td>
<td>match t with {Empty ↦ t₁</td>
</tr>
<tr>
<td>A ⊕ B</td>
<td>left t, right t</td>
<td>match t with {left ↦ t₁</td>
</tr>
</tbody>
</table>

Nat ≅ I ⊕ Nat

---

**General inductive types:**

\[ X : A \]

\[ \text{Nat} = X : I \]

\[ \text{List}(A) = X : I(\text{A List}(A)) \]
## Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let * = t in t'</td>
</tr>
<tr>
<td>A → B</td>
<td>λx.t</td>
<td>tt'</td>
</tr>
<tr>
<td>A ⊗ B</td>
<td>⟨t₁, t₂⟩</td>
<td>let ⟨x, y⟩ = t in t'</td>
</tr>
<tr>
<td>Nat</td>
<td>zero, S t</td>
<td>match t with {zero → t₁</td>
</tr>
<tr>
<td>List(A)</td>
<td>Empty, Cons ⟨h, t⟩</td>
<td>match t with {Empty → t₁</td>
</tr>
<tr>
<td>A ⊕ B</td>
<td>left t, right t</td>
<td>match t with {left → t₁</td>
</tr>
</tbody>
</table>

Nat ≅ I ⊕ Nat    List(A) ≅ I ⊕ (A ⊗ List(A))
### Introduction: Type constructors and destructors

<table>
<thead>
<tr>
<th>Types</th>
<th>Constructor</th>
<th>Destructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>*</td>
<td>let ( * = t ) in ( t' )</td>
</tr>
<tr>
<td>( A \rightarrow B )</td>
<td>( \lambda x.t )</td>
<td>( tt' )</td>
</tr>
<tr>
<td>( A \otimes B )</td>
<td>( \langle t_1, t_2 \rangle )</td>
<td>let ( \langle x, y \rangle = t ) in ( t' )</td>
</tr>
<tr>
<td>Nat</td>
<td>zero, S ( t )</td>
<td>match ( t ) with ( { \text{zero} \mapsto t_1 \mid S \mapsto t_2 } )</td>
</tr>
<tr>
<td>List(( A ))</td>
<td>Empty, Cons ( \langle h, t \rangle )</td>
<td>match ( t ) with ( { \text{Empty} \mapsto t_1 \mid \text{Cons} \mapsto t_2 } )</td>
</tr>
<tr>
<td>( A \oplus B )</td>
<td>left ( t ), right ( t )</td>
<td>match ( t ) with ( { \text{left} \mapsto t_1 \mid \text{right} \mapsto t_2 } )</td>
</tr>
</tbody>
</table>

\[ \text{Nat} \cong I \oplus \text{Nat} \quad \text{List}(A) \cong I \oplus (A \otimes \text{List}(A)) \]

**General** inductive types: \( \mu X.A. \)

\[ \text{Nat} = \mu X.I \oplus X \quad \text{List}(A) = \mu X.I \oplus (A \times X) \]
Introduction: Effects

Dichotomy between purely functional programs and an interaction with the environment.
Introduction: Effects

Dichotomy between purely functional programs and an interaction with the environment.

Effectful types are written with $\mathcal{T}X$.

$$\text{do } x \leftarrow t; t'$$
Introduction: Effects

Dichotomy between purely functional programs and an interaction with the environment.

Effectful types are written with $\mathcal{T}X$.

$$\text{do } x \leftarrow t; t'$$

**Writer** effect

- a tape external to the program
- effectful programs write on the tape
Introduction: Effects

Dichotomy between purely functional programs and an interaction with the environment.

Effectful types are written with $\mathcal{T}X$.

\[
\text{do } x \leftarrow t; t'
\]

**Writer** effect

- a tape external to the program
- effectful programs write on the tape

**Non-deterministic** effect

- Example: $x \leftarrow \text{True} \parallel \text{False}$.
- No control over the value.
Introduction: Effects

Dichotomy between purely functional programs and an interaction with the environment.

Effectful types are written with $TX$.

\[
\text{do } x \leftarrow t; \ t'
\]

**Writer effect**
- a tape external to the program
- effectful programs write on the tape

**Non-deterministic effect**
- Example: $x \leftarrow \text{True} \parallel \text{False}$.
- No control over the value.

**Probabilistic effect**
- Example: $x \leftarrow \text{Coin()}$.
- According to a probability distribution.
Introduction: Effects

Dichotomy between purely functional programs and an interaction with the environment.

Effectful types are written with $\mathcal{T}X$.

$$\text{do } x \leftarrow t; \ t'$$

**Writer** effect
- a tape external to the program
- effectful programs write on the tape

**Probabilistic** effect
- Example: $x \leftarrow \text{Coin()}$.
- According to a probability distribution.

**Non-deterministic** effect
- Example: $x \leftarrow \text{True} \ || \ \text{False}$.
- No control over the value.

**Quantum** effect
- Example: $x \leftarrow \text{QCoin()}$.
- Superposition.
- Can be observed with measurement.
- Quantum bits in $\mathbb{C} \oplus \mathbb{C}$, we write $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.
- Generally: quantum data as normalised vectors in a Hilbert space.
Introduction: Quantum Computing

- Quantum bits in $\mathbb{C} \oplus \mathbb{C}$, we write $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

- Generally: quantum data as normalised vectors in a Hilbert space.

- Physically admissible operations are unitaries (reversible) and measurement (irreversible).
Quantum bits in $\mathbb{C} \oplus \mathbb{C}$, we write $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Generally: quantum data as normalised vectors in a Hilbert space.

Physically admissible operations are unitaries (reversible) and measurement (irreversible).

The most used model of quantum computation is circuits.
Introduction: Quantum Computing

- Quantum bits in $\mathbb{C} \oplus \mathbb{C}$, we write $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.
- Generally: quantum data as normalised vectors in a Hilbert space.
- Physically admissible operations are unitaries (reversible) and measurement (irreversible).
- The most used model of quantum computation is circuits.
Introduction: Quantum Computing

- Quantum bits in $\mathbb{C} \oplus \mathbb{C}$, we write $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.
- Generally: quantum data as normalised vectors in a Hilbert space.
- Physically admissible operations are unitaries (reversible) and measurement (irreversible).
- The most used model of quantum computation is circuits.

Input: $|x, y\rangle$

Program: $x \leftarrow Hx$

$y \leftarrow Ty$

if $x$ then $\langle x, 1 - y\rangle$
else $\langle x, y\rangle$

meas($y$)
Quantum Computing

- Quantum bits in $\mathbb{C} \oplus \mathbb{C}$, we write $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.
- Generally: quantum data as normalized vectors in a Hilbert space.
- Physically admissible operations are unitaries (reversible) and measurement (irreversible).
- The most used model of quantum computation is circuits.

```
Input: |x, y\rangle
Program: x \leftarrow Hx
         y \leftarrow Ty
         if x then \langle x, 1 - y\rangle
         else \langle x, y\rangle
         meas(y)
```
**Quantum $\lambda$-calculus** [SelingerValiron09].

Input: $|x, y\rangle$

Program: $x \leftarrow U_1 x$

  if meas$(x)$  then $U_2 y$

  else $y$

meas$(x)$ is 0 or 1 with a certain probability. Therefore, compute $U_2 y$ is only probable.
Quantum $\lambda$-calculus \cite{SelingerValiron09}.

Input: $|x, y\rangle$

Program: $x \leftarrow U_1 x$

if $\text{meas}(x)$ then $U_2 y$
else $y$

$\text{meas}(x)$ is 0 or 1 with a certain probability.
Therefore, compute $U_2 y$ is only probable.

Our goal (Chapter 3).

Input: $|x, y\rangle$

Program: $x \leftarrow U_1 x$

if $x$ then $|x, U_2 y\rangle$
else $|x, y\rangle$

If $x$ is a superposition $\alpha |0\rangle + \beta |1\rangle$, we get

$$\alpha |0, y\rangle + \beta |1, U_2 y\rangle.$$
**Quantum $\lambda$-calculus** [SelingerValiron09].

Input: $|x, y\rangle$

Program: $x \leftarrow U_1 x$

if $\text{meas}(x)$ then $U_2 y$
else $y$

$\text{meas}(x)$ is 0 or 1 with a certain probability.
Therefore, **compute** $U_2 y$ is only probable.

Studied with a point of view from semantics.

**Our goal** (Chapter 3).

Input: $|x, y\rangle$

Program: $x \leftarrow U_1 x$

if $x$ then $|x, U_2 y\rangle$
else $|x, y\rangle$

If $x$ is a superposition $\alpha |0\rangle + \beta |1\rangle$, we get

$$\alpha |0, y\rangle + \beta |1, U_2 y\rangle.$$
Introduction: Why study semantics?

Comes from Ancient Greek, to provide a meaning.
Introduction: Why study semantics?

Comes from Ancient Greek, *to provide a meaning*.

Operational semantics: **formal** program execution $t \rightarrow t'$.
Introduction: Why study semantics?

Comes from Ancient Greek, *to provide a meaning*.

Operational semantics: *formal* program execution $t \rightarrow t'$.

For example, $(\lambda x. x + 2)3 \rightarrow$
Introduction: Why study semantics?

Comes from Ancient Greek, to provide a meaning.

Operational semantics: formal program execution $t \rightarrow t'$.

For example, $(\lambda x. x + 2)3 \rightarrow 3 + 2 \rightarrow$
Introduction: Why study semantics?

Comes from Ancient Greek, *to provide a meaning*.

Operational semantics: *formal* program execution $t \rightarrow t'$.

For example, $(\lambda x.x + 2)3 \rightarrow 3 + 2 \rightarrow 5$. 
Introduction: Why study semantics?

Comes from Ancient Greek, *to provide a meaning*.

Operational semantics: *formal* program execution $t \rightarrow t'$.

For example, $(\lambda x. x + 2)3 \rightarrow 3 + 2 \rightarrow 5$.

Denotational semantics: mathematical *function* $[t]$.
Introduction: Why study semantics?

Comes from Ancient Greek, to provide a meaning.

Operational semantics: formal program execution \( t \rightarrow t' \).

For example, \((\lambda x.x + 2)3 \rightarrow 3 + 2 \rightarrow 5\).

Denotational semantics: mathematical function \([t]\).

Strong enough if relation between operational and denotational:

\[ t \simeq t' \iff [t] = [t'] \]
Introduction: Why study semantics?

Comes from Ancient Greek, *to provide a meaning*.

Operational semantics: formal program execution $t \rightarrow t'$.

For example, $(\lambda x.x + 2)3 \rightarrow 3 + 2 \rightarrow 5$.

Denotational semantics: mathematical function $[[t]]$.

Strong enough if relation between operational and denotational:

$$ t \simeq t' \iff [[t]] = [[t']] $$

- Prove that the language does what it is supposed to do,
- Compile the language and perform optimisations *safely*,
- Inform on which *features* can be added to the language.
**Introduction: Category theory**

**Category**
- Objects: $X, Y, Z, \ldots$
- Morphisms: $f: X \to Y$.
- Composition:
  \[
  \frac{f: X \to Y \quad g: Y \to Z}{g \circ f: X \to Z}
  \]
- Identity morphism: $\text{id}_X: X \to X$.

**Type system**
- Types: $A, B, C, \ldots$
- Type judgements: $x: A \vdash t: B$.
- Also composable:
  \[
  \frac{x: A \vdash t: B \quad y: B \vdash t': C}{x: A \vdash t'[t/y]: C}
  \]
- Identity axiom: $x: A \vdash x: A$. 

**Most effects are interpreted as a (strong) monad** [Moggi89].

Example (writer effect), with a monoid $(M; m; e)$:
- Monad: $M$
- Objects: $M X$
- Unit: $X = \{ X \mapsto \{(e; x)\} \}$
- Multiplication:
  \[
  X = \{ M(M X) \mapsto \{(m; (m'; x))\} \}
  \]
- Identity axiom: $x: A \vdash x: A$. 

[7]
Introduction: Category theory

Category

- **Objects**: $X, Y, Z, \ldots$
- **Morphisms**: $f: X \to Y$.
- **Composition**:

$$f: X \to Y \quad g: Y \to Z \quad \Rightarrow \quad g \circ f: X \to Z$$

- **Identity morphism**: $\text{id}_X: X \to X$.

Most effects are interpreted as a (strong) monad $\mathcal{T}$ [Moggi89].

Type system

- **Types**: $A, B, C, \ldots$
- **Type judgements**: $x: A \vdash t: B$.
- **Also composable**:

$$x: A \vdash t: B \quad y: B \vdash t': C \quad \Rightarrow \quad x: A \vdash t'[t/y]: C$$

- **Identity axiom**: $x: A \vdash x: A$. 
Introduction: Category theory

Category
- Objects: $X, Y, Z, \ldots$
- Morphisms: $f: X \rightarrow Y$
- Composition:

\[
\frac{f: X \rightarrow Y \quad g: Y \rightarrow Z}{g \circ f: X \rightarrow Z}
\]
- Identity morphism: $\text{id}_X: X \rightarrow X$

Type system
- Types: $A, B, C, \ldots$
- Type judgements: $x: A \vdash t: B$
- Also composable:

\[
\frac{x: A \vdash t: B \quad y: B \vdash t': C}{x: A \vdash t'[t/y]: C}
\]
- Identity axiom: $x: A \vdash x: A$

Most effects are interpreted as a (strong) monad $\mathcal{T}$ [Moggi89].

Example (writer effect), with a monoid $(M, \cdot, e)$:
- Monad: $M \times -$  
- Objects: $M \times X$
Introduction: Category theory

Category

- Objects: $X, Y, Z, \ldots$
- Morphisms: $f: X \rightarrow Y$.
- Composition:

$$
\frac{f: X \rightarrow Y \quad g: Y \rightarrow Z}{g \circ f: X \rightarrow Z}
$$

- Identity morphism: $\text{id}_X: X \rightarrow X$.

Type system

- Types: $A, B, C, \ldots$
- Type judgements: $x: A \vdash t: B$.
- Also composable:

$$
\frac{x: A \vdash t: B \quad y: B \vdash t': C}{x: A \vdash t'[t/y]: C}
$$

- Identity axiom: $x: A \vdash x: A$.

Most effects are interpreted as a (strong) monad $\mathcal{T}$ [Moggi89].

Example (writer effect), with a monoid $(M, \cdot, e)$:  

Monad: $M \times -$  
Objects: $M \times X$

Unit: $\eta_X = \left\{ \begin{array}{ll} 
X & \rightarrow M \times X \\
\, \, \, \, x & \mapsto (e, x) \end{array} \right.$
Introduction: Category theory

Category

- Objects: $X, Y, Z, \ldots$
- Morphisms: $f: X \to Y$.
- Composition:

$$f: X \to Y \quad g: Y \to Z$$
$$\quad \Rightarrow g \circ f: X \to Z$$

- Identity morphism: $\text{id}_X: X \to X$.

Type system

- Types: $A, B, C, \ldots$
- Type judgements: $x: A \vdash t: B$.
- Also composable:

$$x: A \vdash t: B \quad y: B \vdash t': C$$
$$\quad \Rightarrow x: A \vdash t'[t/y]: C$$

- Identity axiom: $x: A \vdash x: A$.

Most effects are interpreted as a (strong) monad $T$ [Moggi89].

Example (writer effect), with a monoid $(M, \cdot, e)$:

Monad: $M \times -$  
Objects: $M \times X$

Unit: $\eta_X = \begin{cases} X & \rightarrow & M \times X \\ x & \mapsto & (e, x) \end{cases}$

Multiplication: $\mu_X = \begin{cases} M \times (M \times X) & \rightarrow & M \times X \\ (m, (m', x)) & \mapsto & (m \cdot m', x) \end{cases}$
Contents

- Effects
  - Centrality
    - Chapter 2, LICS’23
  - Quantum Control
    - Chapter 3, Submitted
  - Reversible Recursion
    - Chapter 4, FSCD’24
  - Quantum Recursion
    - Chapter 5
Centrality: Motivation

- **Monads** represent computational *effects*.
- Monads have an algebraic flavour (category theory).
- Centre (algebraically): elements that *commute* with all others.
Centrality: Motivation

- **Monads** represent computational **effects**.
- Monads have an algebraic flavour (category theory).
- Centre (algebraically): elements that **commute** with all others.

---

Two examples of monadic sequencing in Haskell.

```haskell
p1:
do
  x <- op1
  y <- op2
  f x y

p2:
do
  y <- op2
  x <- op1
  f x y
```

---

**Intuition**: If `op1` or `op2` is **central**, `p1` and `p2` should be equivalent.
- Monads represent computational effects.
- Monads have an algebraic flavour (category theory).
- Centre (algebraically): elements that commute with all others.

```
  p1: p2:
  do   do
    x <- op1    y <- op2
    y <- op2    x <- op1
    f x y      f x y
```

Two examples of monadic sequencing in Haskell.

- Intuition: If \( \text{op1} \) or \( \text{op2} \) is central, \( p1 \) and \( p2 \) should be equivalent.

Question:

What is centrality for monads?
Centrality: Background on Premonoidal categories

[PowerRobinson97] Premonoidal categories as model of effects.

A premonoidal category $\mathcal{P}$ has a centre $Z(\mathcal{P})$;
$Z(\mathcal{P})$ is a monoidal subcategory of $\mathcal{P}$. 
**Centrality: Strong monads**

**Strong** monad $\mathcal{T}$: combines effect with pairing ($\otimes$) with a strength $\tau$:

$$\tau_{X,Y} : X \otimes \mathcal{T}Y \rightarrow \mathcal{T}(X \otimes Y).$$

Operationally:

- **Input:** $\langle x, M \rangle$
- **Output:** `do y <- M ; return \langle x, y \rangle`

With strong monad $\mathcal{T} : \mathbf{C} \rightarrow \mathbf{C}$, the category $\mathbf{C}_\mathcal{T}$:

- Objects of $\mathbf{C}$;
- Morphisms $X \rightarrow \mathcal{T}Y$.

is called Kleisli category, and is a premonoidal category.
Centrality: Examples on Set

**Writer monad**
- Monoid $M$ with centre $Z(M)$.
- Monad $\rightarrow (M \times -) : \mathbf{Set} \rightarrow \mathbf{Set}$.
- Centre $\rightarrow (Z(M) \times -) : \mathbf{Set} \rightarrow \mathbf{Set}$.

**Powerset monad: non-determinism**
- Commutative.
- Centre $\rightarrow$ itself.
**Centrality: Examples on Set**

**Writer monad**
- Monoid $M$ with centre $Z(M)$.
- Monad – $(M \times -) : \text{Set} \to \text{Set}.$
- Centre – $(Z(M) \times -) : \text{Set} \to \text{Set}.$

Obtained by considering the appropriate subset $\mathcal{Z}X \subseteq \mathcal{T}X.$

**Powerset monad: non-determinism**
- Commutative.
- Centre – itself.
**Centrality: Central cone**

**Commutative monad:**
for all $M$ and $N$:

\[
\begin{align*}
  x &\leftarrow N \\
y &\leftarrow M \\
\text{return } \langle x, y \rangle
\end{align*}
\]
\[
\begin{align*}
  y &\leftarrow M \\
x &\leftarrow N \\
\text{return } \langle x, y \rangle
\end{align*}
\]

**Central cone:** A pair $(Z, \iota : Z \to TX)$ such that, for all $M$:

\[
\begin{align*}
  x &\leftarrow \iota(z) \\
y &\leftarrow M \\
\text{return } \langle x, y \rangle
\end{align*}
\]
\[
\begin{align*}
  y &\leftarrow M \\
x &\leftarrow \iota(z) \\
\text{return } \langle x, y \rangle
\end{align*}
\]
Centrality: Central cone

Commutative monad:
for all $M$ and $N$:

\[
\begin{align*}
x & \leftarrow N \\
y & \leftarrow M \\
\text{return } \langle x, y \rangle
\end{align*}
\]

\[
\begin{align*}
x & \leftarrow M \\
y & \leftarrow N \\
\text{return } \langle x, y \rangle
\end{align*}
\]

Central cone: A pair $(Z, \iota : Z \to TX)$
such that, for all $M$:

\[
\begin{align*}
x & \leftarrow \iota(z) \\
y & \leftarrow M \\
\text{return } \langle x, y \rangle
\end{align*}
\]

\[
\begin{align*}
x & \leftarrow M \\
y & \leftarrow \iota(z) \\
\text{return } \langle x, y \rangle
\end{align*}
\]

If the universal central cone at $X$ exists, write $\mathcal{Z}X \overset{\text{def}}{=} Z$. 
**Theorem 2.11**: Equivalent conditions for a monad $\mathcal{T}$ to have a centre:

1. All universal central cones.
2. A monad $\mathcal{Z}$ linked to $Z(\mathcal{C}_\mathcal{T})$.
3. An adjunction between $\mathcal{C}$ and $Z(\mathcal{C}_\mathcal{T})$. 

**Corollaries**

1. Strong monads on $\text{Set}$; $\text{DCPO}$; $\text{Top}$; $\text{Vect}$;::: are centralisable.
2. A commutative monad is its own centre.
3. If $\mathcal{C}$ closed and total, every strong monad on it admits a centre.

All strong monads centralisable? No, but only artificial counterexamples! 

If $\mathcal{M}$ monoid, $\mathcal{X}$ $\mathcal{Z}(\mathcal{M})$ contains only central elements.

Central submonad: $\mathcal{S}$, !$\mathcal{T}$ with only central effects.
Theorem 2.11: Equivalent conditions for a monad $\mathcal{T}$ to have a centre:

1. All universal central cones.
2. A monad $\mathcal{Z}$ linked to $\mathcal{Z}(\mathcal{C}_\mathcal{T})$.
3. An adjunction between $\mathcal{C}$ and $\mathcal{Z}(\mathcal{C}_\mathcal{T})$.

Corollaries (1) Strong monads on $\textbf{Set}$, $\textbf{DCPO}$, $\textbf{Top}$, $\textbf{Vect}$, ... are centralisable.
Theorem 2.11: Equivalent conditions for a monad \( T \) to have a centre:

1. All universal central cones.
2. A monad \( Z \) linked to \( Z(C_T) \).
3. An adjunction between \( C \) and \( Z(C_T) \).

Corollaries

1. Strong monads on \( \text{Set, DCPO, Top, Vect, \ldots} \) are centralisable.
2. A commutative monad is its own centre.
Theorem 2.11: Equivalent conditions for a monad $\mathcal{T}$ to have a centre:

1. All universal central cones.
2. A monad $\mathcal{Z}$ linked to $Z(\mathcal{C}_\mathcal{T})$.
3. An adjunction between $\mathcal{C}$ and $Z(\mathcal{C}_\mathcal{T})$.

Corollaries

1. Strong monads on $\text{Set}$, $\text{DCPO}$, $\text{Top}$, $\text{Vect}$, $\ldots$ are centralisable.
2. A commutative monad is its own centre.
3. If $\mathcal{C}$ closed and total, every strong monad on it admits a centre.
**Theorem 2.11**: Equivalent conditions for a monad $\mathcal{T}$ to have a centre:

1. All universal central cones.
2. A monad $\mathcal{Z}$ linked to $\mathcal{Z}(\mathcal{C}_T)$.
3. An adjunction between $\mathcal{C}$ and $\mathcal{Z}(\mathcal{C}_T)$.

**Corollaries**

1. Strong monads on $\textbf{Set}$, $\textbf{DCPO}$, $\textbf{Top}$, $\textbf{Vect}$, ... are centralisable.
2. A commutative monad is its own centre.
3. If $\mathcal{C}$ closed and total, every strong monad on it admits a centre.

All strong monads centralisable?
Theorem 2.11: Equivalent conditions for a monad $\mathcal{T}$ to have a centre:

1. All universal central cones.
2. A monad $\mathcal{Z}$ linked to $Z(\mathcal{C}_\mathcal{T})$.
3. An adjunction between $\mathcal{C}$ and $Z(\mathcal{C}_\mathcal{T})$.

Corollaries

1. Strong monads on $\text{Set}$, $\text{DCPO}$, $\text{Top}$, $\text{Vect}$, $\ldots$ are centralisable.
2. A commutative monad is its own centre.
3. If $\mathcal{C}$ closed and total, every strong monad on it admits a centre.

All strong monads centralisable? No, but only artificial counterexamples!
Theorem 2.11: Equivalent conditions for a monad $T$ to have a centre:

1. All universal central cones.
2. A monad $Z$ linked to $Z(C_T)$.
3. An adjunction between $C$ and $Z(C_T)$.

Corollaries

1. Strong monads on $\textbf{Set, DCPO, Top, Vect, \ldots}$ are centralisable.
2. A commutative monad is its own centre.
3. If $C$ closed and total, every strong monad on it admits a centre.

All strong monads centralisable? No, but only artificial counterexamples!

If $M$ monoid, $X \subseteq Z(M)$ contains only central elements.
Theorem 2.11: Equivalent conditions for a monad $\mathcal{T}$ to have a centre:

1. All universal central cones.
2. A monad $\mathcal{Z}$ linked to $\mathcal{Z}(\mathcal{C}_\mathcal{T})$.
3. An adjunction between $\mathcal{C}$ and $\mathcal{Z}(\mathcal{C}_\mathcal{T})$.

Corollaries

1. Strong monads on $\text{Set}$, $\text{DCPO}$, $\text{Top}$, $\text{Vect}$, $\ldots$ are centralisable.
2. A commutative monad is its own centre.
3. If $\mathcal{C}$ closed and total, every strong monad on it admits a centre.

All strong monads centralisable? No, but only artificial counterexamples!

If $M$ monoid, $X \subseteq Z(M)$ contains only central elements.

Central submonad: $S \hookrightarrow \mathcal{T}$ with only central effects.
<table>
<thead>
<tr>
<th>Language</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simply-typed $\lambda$-calculus (ST$\lambda$C)</td>
<td>Cartesian Closed Category (CCC)</td>
</tr>
<tr>
<td>Moggi's metalanguage</td>
<td>CCC with strong monad $T$ [Moggi89]</td>
</tr>
<tr>
<td>???</td>
<td>CCC with central submonad $S \twoheadrightarrow T$</td>
</tr>
</tbody>
</table>
**Centrality: Computational Interpretation**

<table>
<thead>
<tr>
<th>Language</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simply-typed $\lambda$-calculus (ST$\lambda$C)</td>
<td>Cartesian Closed Category (CCC)</td>
</tr>
<tr>
<td>Moggi's metalanguage</td>
<td>CCC with strong monad $\mathcal{T}$ [Moggi89]</td>
</tr>
<tr>
<td>CSC</td>
<td>CCC with central submonad $S \hookrightarrow \mathcal{T}$</td>
</tr>
</tbody>
</table>

**CSC (Central Submonad Calculus):**

- Simply-typed $\lambda$-calculus;
- new types: $SX$ and $TX$;
- terms for monadic computation (à la Moggi);
- equational rules, such as:

\[
\Gamma \vdash \text{do}_{\mathcal{T}} x \leftarrow \nu M; \text{do}_{\mathcal{T}} y \leftarrow N; \ P = \text{do}_{\mathcal{T}} y \leftarrow N; \text{do}_{\mathcal{T}} x \leftarrow \nu M; \ P : \mathcal{T} C
\]
Centrality: Completeness and Internal Language

**Folklore:**

Completeness

ST\(\lambda\)Cs \(\overset{\text{R}}{\rightarrow}\) CCCs

Internal Language

**Theorem 2.60**

Completeness

CSCs \(\overset{\text{R}}{\rightarrow}\) CSC-models

Internal Language
Usual quantum programming languages: unitaries (circuits) and measurement.
Quantum Control: Motivation

Usual quantum programming languages: unitaries (circuits) and measurement.

Example, Toffoli:
Quantum Control: Motivation

Usual quantum programming languages: unitaries (circuits) and measurement.

Example, Toffoli:

\[
\begin{bmatrix}
|0\rangle \otimes y \otimes z & \mapsto & |0\rangle \otimes y \otimes z \\
|x \otimes |0\rangle \otimes z & \mapsto & x \otimes |0\rangle \otimes z \\
|1\rangle \otimes |1\rangle \otimes z & \mapsto & |1\rangle \otimes |1\rangle \otimes (1 - z)
\end{bmatrix}
\]
Usual quantum programming languages: unitaries (circuits) and measurement.

Example, Toffoli:

\[
\begin{align*}
|0\rangle \otimes y \otimes z & \quad \mapsto \quad |0\rangle \otimes y \otimes z \\
|1\rangle \otimes |0\rangle \otimes z & \quad \mapsto \quad x \otimes |0\rangle \otimes z \\
|1\rangle \otimes |1\rangle \otimes z & \quad \mapsto \quad |1\rangle \otimes |1\rangle \otimes (1 - z)
\end{align*}
\]

Goal:

Represent syntactically normalised quantum states and unitaries. Based on [SVV18].
Quantum Control: Motivation

Usual quantum programming languages: unitaries (circuits) and measurement.

Example, Toffoli:

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

Goal:

Represent syntactically normalised quantum states and unitaries. Based on [SVV18].

Difficulties:

- A notion of orthogonality for the norm.
- A notion of orthonormal basis for unitarity.
Quantum Control: The syntax

Types:

I \quad A \oplus B \quad A \otimes B \quad \text{Nat}

Terms and semantics as isometries:
Quantum Control: The syntax

Types:

\[
\begin{array}{cccc}
I & A \oplus B & A \otimes B & Nat \\
\end{array}
\]

Terms and semantics as isometries:

Syntax \quad \mathbb{C}^n

\[
\begin{align*}
*: I & \rightarrow [1] \quad 1 \\
\text{left} *: I \oplus I & \rightarrow [1] \quad |0\rangle \\
\text{right} *: I \oplus I & \rightarrow [0] \quad |1\rangle \\
(\alpha \cdot \text{left} *) + (\beta \cdot \text{right} *): I \oplus I & \rightarrow \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \alpha |0\rangle + \beta |1\rangle
\end{align*}
\]
Quantum Control: The syntax

Types:

\[ I \quad A \oplus B \quad \boxed{A \otimes B} \quad \text{Nat} \]

Terms and semantics as isometries:

\[ t_i : I \oplus I \quad \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \quad |\phi_i\rangle \]

\[ t_1 \otimes t_2 : (I \oplus I) \otimes (I \oplus I) \quad \begin{bmatrix} \alpha_1 \alpha_2 \\ \beta_1 \alpha_2 \\ \alpha_1 \beta_2 \\ \beta_1 \beta_2 \end{bmatrix} \quad |\phi_1\rangle \otimes |\phi_2\rangle \]
Quantum Control: The syntax

Types:

\[ \text{I} \quad A \oplus B \quad A \otimes B \quad \text{Nat} \]

Terms and semantics as isometries:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>$\ell^2(\mathbb{N})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero: Nat</td>
<td>$t \begin{bmatrix} 1 &amp; 0 &amp; \cdots \end{bmatrix}$</td>
</tr>
<tr>
<td>S zero: Nat</td>
<td>$t \begin{bmatrix} 0 &amp; 1 &amp; 0 &amp; \cdots \end{bmatrix}$</td>
</tr>
<tr>
<td>$(\alpha \cdot \text{zero}) + (\beta \cdot S \ S \text{zero}): \text{Nat}$</td>
<td>$t \begin{bmatrix} \alpha &amp; 0 &amp; \beta &amp; 0 &amp; \cdots \end{bmatrix}$</td>
</tr>
</tbody>
</table>
Quantum Control: Examples of Unitaries

Can be seen as \( \lambda \)-abstractions with pattern-matching.

\[
\begin{align*}
\text{id} &= \{ x \mapsto x \}: \text{qubit} \to \text{qubit} \\
\text{NOT} &= \begin{cases} 
ff \mapsto tt \\
tt \mapsto ff
\end{cases}: \text{qubit} \to \text{qubit} \\
\text{QCoin} &= \begin{cases} 
ff \mapsto \frac{1}{\sqrt{2}}ff + \frac{1}{\sqrt{2}}tt \\
tt \mapsto \frac{1}{\sqrt{2}}ff - \frac{1}{\sqrt{2}}tt
\end{cases}: \text{qubit} \to \text{qubit} \\
\text{SWAP} &= \{ x \otimes y \mapsto y \otimes x \}: \text{qubit}^{\otimes 2} \to \text{qubit}^{\otimes 2} \\
\text{CNOT} &= \begin{cases} 
(ff) \otimes x \mapsto (ff) \otimes x \\
(tt) \otimes (ff) \mapsto (tt) \otimes (tt) \\
(tt) \otimes (tt) \mapsto (tt) \otimes (ff)
\end{cases}: \text{qubit}^{\otimes 2} \to \text{qubit}^{\otimes 2}
\end{align*}
\]

\[\begin{aligned}
tt &= \text{right} * \\
ff &= \text{left} * \\
\text{qubit} &= I \oplus I \\
\text{qubit}^{\otimes 2} &= \text{qubit} \otimes \text{qubit}
\end{aligned}\]
Quantum Control: Orthogonality

Mathematically (Hilbert spaces):
- Normalised vector $|\phi\rangle$.
- Orthogonality: $|\phi\rangle \perp |\psi\rangle$.
- Orthonormal basis (ONB): $\{|\phi_i\rangle\}_i$.
- Normalised superposition:
  
  If $|\phi_i\rangle$ are pairwise orthogonal, and $\sum_i |\alpha_i|^2 = 1$, then $\sum_i \alpha_i |\phi_i\rangle$ is normalised.

Syntactically:
- Well-typed term $\Delta \vdash t : A$.
- Orthogonality relation: $t_1 \perp t_2$.
- Orthogonal decomposition: $\text{OD}_A(\{t_i\}_i)$.
- Typed superposition:
  
  $\Delta \vdash t_i : A \quad t_j \perp t_k \quad \sum_i |\alpha_i|^2 = 1$

  $\Delta \vdash \sum_i (\alpha_i \cdot t_i) : A$
Mathematically (Hilbert spaces):

- Unitaries.
- Orthonormal basis (ONB): \{|\phi_i\rangle\}_i.
- Bijection between two ONBs $\leftrightarrow$ unitary.
- Resolution of the identity.

Syntactically:

- Syntactic unitaries.
- Orthogonal decomposition: $\text{OD}_A(\{t_i\}_i)$.

\[
\begin{align*}
\text{OD}_A(\{t_i\}_i) & \quad \text{OD}_B(\{t'_i\}_i) \\
\Delta_i \vdash t_i : A & \quad \Delta_i \vdash t'_i : B \\
\{t_i \mapsto t'_i\} : A \rightarrow B
\end{align*}
\]

- Denotational resolution of the identity.
Quantum Control: Completeness

Equational theory
(Figures 3.5-8)

Completeness
(Theorem 3.67)

Denotational semantics
(Hilbert spaces and isometries)

\[ \vdash t_1 = t_2 : A \iff [\cdot \vdash t_1 : A] = [\cdot \vdash t_2 : A] \]

Unique normal form
(Lemma 3.33)
Contents

Effects

Centrality
*Chapter 2*

Quantum Control
*Chapter 3*

Reversible Recursion
*Chapter 4*

Quantum Recursion
*Chapter 5*
Reversible Recursion: Motivation

Question:

How does reversibility interact with infinite data?
Reversible Recursion: Motivation

Question:

How does reversibility interact with infinite data?

In Chapter 4:

- Focus on classical reversible fragment.
- Inductive types.
- A fixed point operator.

[KaarsgaardRennela21]
Inverse category:

- Every morphism $f$ has a partial inverse.
- Every morphism $f$ has a domain $\tilde{f}$.
- $f \leq g$ iff $f = g\tilde{f}$. 
Reversible Recursion: Mathematical model

**Inverse category:**
- Every morphism \( f \) has a **partial** inverse.
- Every morphism \( f \) has a **domain** \( \bar{f} \).
- \( f \preceq g \) iff \( f = g\bar{f} \).

**With a join structure:**
- If \( f, g : X \rightarrow Y \) agree on there domains, then \( f \lor g \).
Inverse category:

- Every morphism $f$ has a partial inverse.
- Every morphism $f$ has a domain $\bar{f}$.
- $f \leq g$ iff $f = g\bar{f}$.

With a join structure:

If $f, g : X \to Y$ agree on their domains, then $f \vee g$.

Together with function variables and a fixed point operator.
Reversible Recursion: Mathematical model

Inverse category:
- Every morphism $f$ has a partial inverse.
- Every morphism $f$ has a domain $\tilde{f}$.
- $f \leq g$ iff $f = g\tilde{f}$.

With a join structure:
- If $f, g : X \to Y$ agree on their domains, then $f \lor g$.

Together with function variables and a fixed point operator.

Given two sets $A$ and $B$, the set $[A \rightharpoonup B]$ has nice properties for fixed points.

(A rig join inverse category is enriched in pointed dcpos.)
This language is Turing-complete! (Theorem 4.60, proven by Kostia Chardonnet)
This language is Turing-complete! (Theorem 4.60, proven by Kostia Chardonnet)

**Theorem 4.29:** Sound and adequate denotational semantics.
This language is Turing-complete! (Theorem 4.60, proven by Kostia Chardonnet)

**Theorem 4.29**: Sound and adequate denotational semantics.

Concrete model: partial injective functions.

**Theorem 4.61**: Any computable partial injection is representable in the language.
This language is Turing-complete! (Theorem 4.60, proven by Kostia Chardonnet)

**Theorem 4.29**: Sound and adequate denotational semantics.

Concrete model: *partial injective* functions.

**Theorem 4.61**: Any *computable* partial injection is representable in the language.

However!
Contents

- Centrality
  *Chapter 2*

- Quantum Control
  *Chapter 3*

- Reversible Recursion
  *Chapter 4*

- Quantum Recursion
  *Chapter 5*
The *nice* structure (inherited from partial injections) is not preserved.
The nice structure (inherited from partial injections) is not preserved.

**Assumptions:**

- Partial identity smaller than the identity.
- Order preserved by composition.
Quantum Recursion?

The nice structure (inherited from partial injections) is not preserved.

**Assumptions:**

- Partial identity smaller than the identity.
- Order preserved by composition.

\[
\begin{bmatrix}
1 & 0 \\
0 & 0
\end{bmatrix} \land \begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix},
\]
Quantum Recursion?

The *nice* structure (inherited from partial injections) is not preserved.

**Assumptions:**

- Partial identity smaller than the identity.
- Order preserved by composition.

Thus

\[
\begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\
1 & 1
\end{bmatrix}
\leq
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix},
\]
Quantum Recursion?

The nice structure (inherited from partial injections) is not preserved.

Assumptions:

- Partial identity smaller than the identity.
- Order preserved by composition.

\[
\begin{pmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
0 & 0
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 0
\end{pmatrix}
\leq
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix},
\]

thus

\[
\begin{pmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
0 & 0
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 0
\end{pmatrix}
\leq
\begin{pmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
0 & 0
\end{pmatrix}
\begin{pmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
0 & 0
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
\frac{1}{\sqrt{2}} \\
0
\end{pmatrix},
\]

i.e.

\[
\frac{1}{2} \leq 0.
\]
An attempt: Guarded Recursion

Addition of:

- New types ▶ A.
- New constructor next: pause the computation.
An attempt: Guarded Recursion

Addition of:

- New types $\triangleright A$.
- New constructor $\text{next}$: 
  \text{pause} the computation.

Semantics of guarded recursion [BMSS12]. $X_0 \leftarrow X_1 \leftarrow X_2 \leftarrow X_3 \leftarrow \ldots$

The topos of trees $\mathbf{Set}^\text{Nop}$
An attempt: Guarded Recursion

Addition of:

- New types $\triangleright A$.
- New constructor `next`: pause the computation.

Semantics of guarded recursion [BMSS12]. $X_0 \leftarrow X_1 \leftarrow X_2 \leftarrow X_3 \leftarrow \ldots$

The topos of trees $\mathbf{Set}^{\mathbb{N}_{\text{op}}}$ (CCC $\leftrightarrow \lambda$-calculus).

Results:
An attempt: Guarded Recursion

Addition of:

- New types \( A \).
- New constructor \texttt{next}: pause the computation.

Semantics of guarded recursion [BMSS12]. \( X_0 \leftarrow X_1 \leftarrow X_2 \leftarrow X_3 \leftarrow \ldots \)

The topos of trees \( \text{Set}^{\text{op}} \) (CCC \( \leftrightarrow \lambda \)-calculus).

Results:

- Defined a (non-cartesian) category enriched in \( \text{Set}^{\text{op}} \) (Lemma 5.5).
An attempt: Guarded Recursion

Addition of:

- New types $\Rightarrow A$.
- New constructor `next`: `pause` the computation.

Semantics of guarded recursion [BMSS12]. $X_0 \leftarrow X_1 \leftarrow X_2 \leftarrow X_3 \leftarrow \ldots$

The topos of trees $\mathbf{Set}^{\text{nop}}$ ($\mathbf{CCC} \leftrightarrow \lambda$-calculus).

Results:

- Defined a (non-cartesian) category enriched in $\mathbf{Set}^{\text{nop}}$ (Lemma 5.5).
- Models infinite data types and fixed point operators (Theorems 5.29 & 5.32).
Centrality and Monads, LICS’23

- With: Titouan Carette and Vladimir Zamdzhiev.
- What: Categorical study and computational interpretation of the centrality of effects.
- Why: A more refined view on the (non-)commutativity of effects.
Conclusion

Centrality and Monads, LICS’23
- With: Titouan Carette and Vladimir Zamdzhiev.
- What: Categorical study and computational interpretation of the centrality of effects.
- Why: A more refined view on the (non-)commutativity of effects.

Syntax for Quantum Control, Submitted
- With: Vladimir Zamdzhiev.
- What: Syntax and semantics for pure quantum computation.
- Why: Go beyond quantum circuits, master quantum control.
Conclusion

Centrality and Monads, LICS’23
- With: Titouan Carette and Vladimir Zamdzhiev.
- What: Categorical study and computational interpretation of the centrality of effects.
- Why: A more refined view on the (non-)commutativity of effects.

Syntax for Quantum Control, Submitted
- With: Vladimir Zamdzhiev.
- What: Syntax and semantics for pure quantum computation.
- Why: Go beyond quantum circuits, master quantum control.

Reversible Recursion, FSCD’24
- With: Kostia Chardonnet, Benoît Valiron.
- What: Semantics for a Turing-complete reversible language.
- Why: Develop reversible languages and make a step towards quantum recursion.