Non-Cartesian Guarded Recursion with Daggers

Louis Lemonnier
Oilthigh Dhùn Èideann

THE UNIVERSITY *of* EDINBURGH
informatics

GT LHC, École Polytechnique. 4th June 2025

Is guarded recursion artificial recursion?

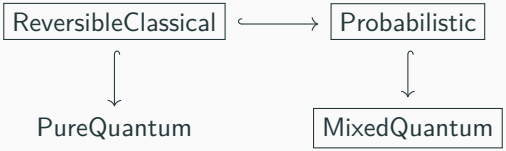
Background

- [Nakano00] defines a modality $\blacktriangleright$, representing the delay between recursive calls.
- Helps for a better formalisation of streams, with notions of productivity.
- [Birkedal&al12] model of guarded recursion in the topos of trees.

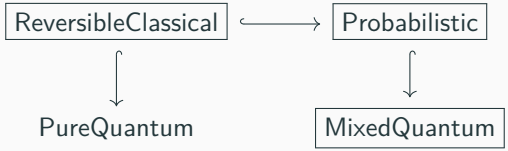
What we do and how

- Obtain (artificial) recursion from any setting.
  - by mimicking the structure of guarded recursion.
- Apply this structure to pure quantum programming (hence the dagger).

Motivation: The quantum troubles with recursion

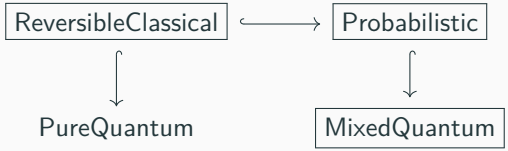


Motivation: The quantum troubles with recursion



Models of reversible, probabilistic and mixed quantum programming have recursion (thanks to enrichment in **DCPO**).

Motivation: The quantum troubles with recursion



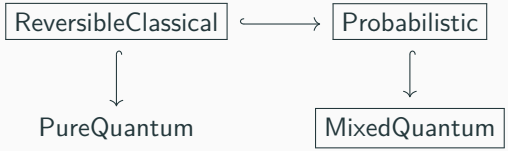
Models of reversible, probabilistic and mixed quantum programming have recursion

(thanks to enrichment in **DCPO**).

Contr (Hilbert spaces and contractive linear maps) is not enriched in an interesting way.

Composition does not preserve any reasonable poset structure.

Motivation: The quantum troubles with recursion



Models of reversible, probabilistic and mixed quantum programming have recursion

(thanks to enrichment in **DCPO**).

Contr (Hilbert spaces and contractive linear maps) is not enriched in an interesting way.

Composition does not preserve any reasonable poset structure.

We propose a solution with techniques adapted from guarded recursion.

Computation in the topos of trees

Objects in the topos of trees $\mathbf{S} = \mathbf{Set}^{\mathbb{N}^{\mathrm{op}}}$ are cochains in **Set**:

$$X(0) \xleftarrow[r_0]{} X(1) \xleftarrow[r_1]{} X(2) \longleftarrow \cdots$$

There is a functor $L\colon \mathbf{S} \to \mathbf{S}$, such that LX is:

$$1 \xleftarrow[!]{} X(0) \xleftarrow[r_0]{} X(1) \xleftarrow[r_1]{} X(2) \longleftarrow \cdots$$

and a natural transformation $\nu\colon \mathrm{id} \Rightarrow L$, such that ν_X is:

$$\begin{array}{ccccccccc} X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \xleftarrow[r_2]{} & X(3) & \longleftarrow & \cdots \\ \downarrow{\scriptstyle !} & & \downarrow{\scriptstyle r_0} & & \downarrow{\scriptstyle r_1} & & \downarrow{\scriptstyle r_2} & & \\ 1 & \xleftarrow[!]{} & X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \longleftarrow & \cdots \end{array}$$

and a family of morphisms $\mathrm{fix}_X\colon [LX \to X] \to X$.

A guarded category

$\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is enriched in the topos of trees $\mathbf{S}$, with hom-objects $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y)$:

- $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y)(n)$ is the set of truncated natural transformations $(\alpha_0, \ldots, \alpha_n)$;
- arrows $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y)(n{+}1) \to \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y)(n)$ forget the last component.

If $\mathbf{C}$ has a terminal object 1,

A guarded category

$\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is enriched in the topos of trees $\mathbf{S}$, with hom-objects $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y)$:

- $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y)(n)$ is the set of truncated natural transformations $(\alpha_0, \ldots, \alpha_n)$;
- arrows $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y)(n{+}1) \to \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y)(n)$ forget the last component.

If $\mathbf{C}$ has a terminal object 1, there is a functor $L^{\mathbf{C}} \colon \mathbf{C}^{\mathbb{N}^{\mathrm{op}}} \to \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$, such that LX is:

$$1 \xleftarrow[!]{} X(0) \xleftarrow[r_0]{} X(1) \xleftarrow[r_1]{} X(2) \longleftarrow \cdots$$

and a natural transformation $\nu^{\mathbf{C}} \colon \mathrm{id} \Rightarrow L$, such that ν_X is:

$$\begin{array}{ccccccccc} X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \xleftarrow[r_2]{} & X(3) & \longleftarrow & \cdots \\ \downarrow{!} & & \downarrow{r_0} & & \downarrow{r_1} & & \downarrow{r_2} & & \\ 1 & \xleftarrow[!]{} & X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \longleftarrow & \cdots \end{array}$$

Some results

A locally contractive functor is a functor that involves $L^{\mathbf{C}}$.

Some results

A locally contractive functor is a functor that involves $L^{\mathbf{C}}$.

Theorem

If an endofunctor $T\colon \mathbf{C}^{\mathbb{N}^{\mathrm{op}}} \to \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is locally contractive, then it has a fixed point $\alpha\colon TX \to X$.

Some results

A locally contractive functor is a functor that involves $L^{\mathbf{C}}$.

Theorem

If an endofunctor $T\colon \mathbf{C}^{\mathbb{N}^{\mathrm{op}}} \to \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is locally contractive, then it has a fixed point $\alpha\colon TX \to X$.

In fact, we have all parameterised fixed points for functors $(\mathbf{C}^{\mathbb{N}^{\mathrm{op}}})^k \to \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$.

Some results

A locally contractive functor is a functor that involves $L^{\mathbf{C}}$.

Theorem

If an endofunctor $T\colon \mathbf{C}^{\mathbb{N}^{\mathrm{op}}} \to \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is locally contractive, then it has a fixed point $\alpha\colon TX \to X$.

In fact, we have all parameterised fixed points for functors $(\mathbf{C}^{\mathbb{N}^{\mathrm{op}}})^k \to \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$.

We also have:

$$L^{\mathbf{Set}}\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X, Y) \cong \mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(L^{\mathbf{C}}X, L^{\mathbf{C}}Y)$$

$$\nu^{\mathbf{Set}}_{\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}(X,Y)} \cong L^{\mathbf{C}}_{X,Y}$$

What about programs?

Consider $\mathbf{C}$ is a model of a typed language:

- types A interpreted as objects $[\![A]\!]$;
- programs $\Gamma \vdash M\colon A$ interpreted as morphisms $[\![\Gamma]\!] \to [\![A]\!]$.

What about programs?

Consider $\mathbf{C}$ is a model of a typed language:

- types A interpreted as objects $[\![A]\!]$;
- programs $\Gamma \vdash M\colon A$ interpreted as morphisms $[\![\Gamma]\!] \to [\![A]\!]$.

Our $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is a model of a similar language with guarded recursion!

- modality $\blacktriangleright A$ interpreted as $L^{\mathbf{C}}[\![A]\!]$;
- constructor `next` interpreted as $\nu^{\mathbf{C}}$;
- (co)inductive data types;
- a guarded fixed-point operator.

What about programs?

Consider $\mathbf{C}$ is a model of a typed language:

- types A interpreted as objects $[\![A]\!]$;
- programs $\Gamma \vdash M\colon A$ interpreted as morphisms $[\![\Gamma]\!] \to [\![A]\!]$.

Our $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is a model of a similar language with guarded recursion!

- modality $\blacktriangleright A$ interpreted as $L^{\mathbf{C}}\,[\![A]\!]$;
- constructor `next` interpreted as $\nu^{\mathbf{C}}$;
- (co)inductive data types;
- a guarded fixed-point operator.

Similar because monoidal structures and exponentials are preserved. What else?

What about programs?

Consider $\mathbf{C}$ is a model of a typed language:

- types A interpreted as objects $[\![A]\!]$;
- programs $\Gamma \vdash M\colon A$ interpreted as morphisms $[\![\Gamma]\!] \to [\![A]\!]$.

Our $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is a model of a similar language with guarded recursion!

- modality $\blacktriangleright A$ interpreted as $L^{\mathbf{C}}[\![A]\!]$;
- constructor `next` interpreted as $\nu^{\mathbf{C}}$;
- (co)inductive data types;
- a guarded fixed-point operator.

Similar because monoidal structures and exponentials are preserved. What else?

We could stop there, but

Reversibility

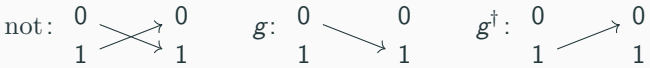
Dagger categories

Origin: functional analysis where $\langle fx \mid y \rangle = \langle x \mid f^\dagger y \rangle$.

Category $\mathbf{C}$ equipped with a functor $(-)^\dagger \colon \mathbf{C}^{\mathrm{op}} \to \mathbf{C}$, such that:

- On objects, $A^\dagger = A$.
- On morphisms: $f^{\dagger\dagger} = f$.

Example with partial injective functions between sets, here $\{0, 1\}$.



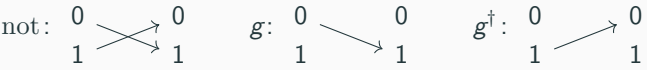
Dagger categories

Origin: functional analysis where $\langle fx \mid y \rangle = \langle x \mid f^\dagger y \rangle$.

Category $\mathbf{C}$ equipped with a functor $(-)^\dagger \colon \mathbf{C}^{\mathrm{op}} \to \mathbf{C}$, such that:

- On objects, $A^\dagger = A$.
- On morphisms: $f^{\dagger\dagger} = f$.

Example with partial injective functions between sets, here $\{0, 1\}$.



If $g^\dagger = g^{-1}$, we say that g is unitary.

Examples of relevant dagger categories

Sets and bijections.

Examples of relevant dagger categories

Sets and bijections.

Sets and partial injections. g:

g is undefined on 1.

Examples of relevant dagger categories

Sets and bijections.

Sets and partial injections. g:

g is undefined on 1.

Hilbert spaces and unitary maps.

$$|0\rangle \longrightarrow |+\rangle$$
$$|1\rangle \longrightarrow |-\rangle$$

where

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad |+\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \qquad |-\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

Examples of relevant dagger categories

Sets and bijections.

Sets and partial injections. g:

g is undefined on 1.

Hilbert spaces and unitary maps.

$$|0\rangle \longrightarrow |+\rangle$$
$$|1\rangle \longrightarrow |-\rangle$$

where

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad |+\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \qquad |-\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

Hilbert spaces and contractions. h:

$$\begin{array}{ccc} |0\rangle & \longrightarrow & |+\rangle \\ |1\rangle & & |-\rangle \end{array}$$

$h|1\rangle = 0.$

The guarded/dagger clash

If $\mathbf{C}$ is a dagger category, $\mathbf{C}^{\mathbb{N}^{\mathrm{OP}}}$ is not necessarily.

The guarded/dagger clash

If $\mathbf{C}$ is a dagger category, $\mathbf{C}^{\mathbb{N}^{\mathrm{OP}}}$ is not necessarily.

E.g. the components of $\nu^{\mathbf{C}}$:

$$\begin{array}{ccccccccc}
X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \xleftarrow[r_2]{} & X(3) & \longleftarrow & \cdots \\
\downarrow{\scriptstyle !} & & \downarrow{\scriptstyle r_0} & & \downarrow{\scriptstyle r_1} & & \downarrow{\scriptstyle r_2} & & \\
1 & \xleftarrow[!]{} & X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \longleftarrow & \cdots
\end{array}$$

The guarded/dagger clash

If $\mathbf{C}$ is a dagger category, $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is not necessarily.

E.g. the components of $\nu^{\mathbf{C}}$:

$$\begin{array}{ccccccccc}
X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \xleftarrow[r_2]{} & X(3) & \longleftarrow & \cdots \\
\downarrow{\scriptstyle !} & & \downarrow{\scriptstyle r_0} & & \downarrow{\scriptstyle r_1} & & \downarrow{\scriptstyle r_2} & & \\
1 & \xleftarrow[!]{} & X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \longleftarrow & \cdots
\end{array}$$

does not yield a dagger:

$$\begin{array}{ccccccccc}
X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \xleftarrow[r_2]{} & X(3) & \longleftarrow & \cdots \\
\uparrow{\scriptstyle !} & & \uparrow{\scriptstyle r_0} & & \uparrow{\scriptstyle r_1} & & \uparrow{\scriptstyle r_2} & & \\
1 & \xleftarrow[!]{} & X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \longleftarrow & \cdots
\end{array}$$

The guarded/dagger clash

If $\mathbf{C}$ is a dagger category, $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ is not necessarily.

E.g. the components of $\nu^{\mathbf{C}}$:

$$\begin{array}{ccccccccc}
X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \xleftarrow[r_2]{} & X(3) & \longleftarrow & \cdots \\
\downarrow{\scriptstyle !} & & \downarrow{\scriptstyle r_0} & & \downarrow{\scriptstyle r_1} & & \downarrow{\scriptstyle r_2} & & \\
1 & \xleftarrow[!]{} & X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \longleftarrow & \cdots
\end{array}$$

does not yield a dagger:

$$\begin{array}{ccccccccc}
X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \xleftarrow[r_2]{} & X(3) & \longleftarrow & \cdots \\
\uparrow{\scriptstyle !} & & \uparrow{\scriptstyle r_0} & & \uparrow{\scriptstyle r_1} & & \uparrow{\scriptstyle r_2} & & \\
1 & \xleftarrow[!]{} & X(0) & \xleftarrow[r_0]{} & X(1) & \xleftarrow[r_1]{} & X(2) & \longleftarrow & \cdots
\end{array}$$

Can we form (interesting) subcategories of $\mathbf{C}^{\mathbb{N}^{\mathrm{op}}}$ that admit a dagger?

Daggerable maps

Definition

$f\colon X \to Y$ *admits a dagger* or is *daggerable* if the family $\{f_n^\dagger\}_{n\in\mathbb{N}}$ verifies:

$$\begin{array}{ccccccccc}
Y(0) & \xleftarrow{r_0^Y} & Y(1) & \xleftarrow{r_1^Y} & Y(2) & \xleftarrow{r_2^Y} & Y(3) & \longleftarrow & \cdots \\
\downarrow{\scriptstyle f_0^\dagger} & & \downarrow{\scriptstyle f_1^\dagger} & & \downarrow{\scriptstyle f_2^\dagger} & & \downarrow{\scriptstyle f_3^\dagger} & & \\
X(0) & \xleftarrow[r_0^X]{} & X(1) & \xleftarrow[r_1^X]{} & X(2) & \xleftarrow[r_2^X]{} & X(3) & \longleftarrow & \cdots
\end{array}$$

Are daggerable:

Daggerable maps

Definition

$f\colon X \to Y$ *admits a dagger* or is *daggerable* if the family $\{f_n^\dagger\}_{n\in\mathbb{N}}$ verifies:

$$\begin{array}{ccccccccc}
Y(0) & \xleftarrow{r_0^Y} & Y(1) & \xleftarrow{r_1^Y} & Y(2) & \xleftarrow{r_2^Y} & Y(3) & \longleftarrow & \cdots \\
\downarrow{\scriptstyle f_0^\dagger} & & \downarrow{\scriptstyle f_1^\dagger} & & \downarrow{\scriptstyle f_2^\dagger} & & \downarrow{\scriptstyle f_3^\dagger} & & \\
X(0) & \xleftarrow[r_0^X]{} & X(1) & \xleftarrow[r_1^X]{} & X(2) & \xleftarrow[r_2^X]{} & X(3) & \longleftarrow & \cdots
\end{array}$$

Are daggerable: morphisms $!_X\colon X \to O$,

Daggerable maps

Definition

$f\colon X \to Y$ *admits a dagger* or is *daggerable* if the family $\{f_n^\dagger\}_{n\in\mathbb{N}}$ verifies:

$$\begin{array}{ccccccccc}
Y(0) & \xleftarrow{r_0^Y} & Y(1) & \xleftarrow{r_1^Y} & Y(2) & \xleftarrow{r_2^Y} & Y(3) & \longleftarrow & \cdots \\
\downarrow{\scriptstyle f_0^\dagger} & & \downarrow{\scriptstyle f_1^\dagger} & & \downarrow{\scriptstyle f_2^\dagger} & & \downarrow{\scriptstyle f_3^\dagger} & & \\
X(0) & \xleftarrow[r_0^X]{} & X(1) & \xleftarrow[r_1^X]{} & X(2) & \xleftarrow[r_2^X]{} & X(3) & \longleftarrow & \cdots
\end{array}$$

Are daggerable: morphisms $!_X\colon X \to O$, monoidal products of daggerable morphisms,

Daggerable maps

Definition

$f\colon X \to Y$ *admits a dagger* or is *daggerable* if the family $\{f_n^\dagger\}_{n\in\mathbb{N}}$ verifies:

$$\begin{array}{ccccccccc}
Y(0) & \xleftarrow{r_0^Y} & Y(1) & \xleftarrow{r_1^Y} & Y(2) & \xleftarrow{r_2^Y} & Y(3) & \longleftarrow & \cdots \\
\downarrow{\scriptstyle f_0^\dagger} & & \downarrow{\scriptstyle f_1^\dagger} & & \downarrow{\scriptstyle f_2^\dagger} & & \downarrow{\scriptstyle f_3^\dagger} & & \\
X(0) & \xleftarrow[r_0^X]{} & X(1) & \xleftarrow[r_1^X]{} & X(2) & \xleftarrow[r_2^X]{} & X(3) & \longleftarrow & \cdots
\end{array}$$

Are daggerable: morphisms $!_X\colon X \to O$, monoidal products of daggerable morphisms, morphisms with only unitary components, and

Daggerable maps

Definition

$f\colon X \to Y$ *admits a dagger* or is *daggerable* if the family $\{f_n^\dagger\}_{n\in\mathbb{N}}$ verifies:

$$\begin{array}{ccccccccc}
Y(0) & \xleftarrow{r_0^Y} & Y(1) & \xleftarrow{r_1^Y} & Y(2) & \xleftarrow{r_2^Y} & Y(3) & \longleftarrow & \cdots \\
\downarrow{\scriptstyle f_0^\dagger} & & \downarrow{\scriptstyle f_1^\dagger} & & \downarrow{\scriptstyle f_2^\dagger} & & \downarrow{\scriptstyle f_3^\dagger} & & \\
X(0) & \xleftarrow[r_0^X]{} & X(1) & \xleftarrow[r_1^X]{} & X(2) & \xleftarrow[r_2^X]{} & X(3) & \longleftarrow & \cdots
\end{array}$$

Are daggerable: morphisms $!_X\colon X \to O$, monoidal products of daggerable morphisms, morphisms with only unitary components, and

$$\nu_Y^{\mathbf{C}} \circ f \circ \left(\nu_X^{\mathbf{C}}\right)^\dagger$$

when f daggerable.

Let's work out some syntax

Reversible syntax: examples and semantics

We mix functions and pattern-matching in a single entity.

- $x \colon A \vdash t \colon C$
- $y \colon B \vdash t' \colon C$
- $t \perp t'$

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & t \\ \mathtt{inj}_r\, y & \mapsto & t' \end{array} \right\} : A \oplus B \leftrightarrow C$$

Reversible syntax: examples and semantics

We mix functions and pattern-matching in a single entity.

- $x \colon A \vdash t \colon C$
- $y \colon B \vdash t' \colon C$
- $t \perp t'$

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & t \\ \mathtt{inj}_r\, y & \mapsto & t' \end{array} \right\} : A \oplus B \leftrightarrow C$$

You can also form: $\left\{ \begin{array}{lcl} t & \mapsto & \mathtt{inj}_l\, x \\ t' & \mapsto & \mathtt{inj}_r\, y \end{array} \right\} : C \leftrightarrow A \oplus B$

Reversible syntax: examples and semantics

We mix functions and pattern-matching in a single entity.

- $x\colon A \vdash t\colon C$
- $y\colon B \vdash t'\colon C$
- $t \perp t'$

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & t \\ \mathtt{inj}_r\, y & \mapsto & t' \end{array} \right\} : A \oplus B \leftrightarrow C$$

You can also form: $\left\{ \begin{array}{lcl} t & \mapsto & \mathtt{inj}_l\, x \\ t' & \mapsto & \mathtt{inj}_r\, y \end{array} \right\} : C \leftrightarrow A \oplus B$

Functions are as general as possible: $\{t_1 \mapsto t'_1 \mid \cdots \mid t_m \mapsto t'_m\}$.

Reversible syntax: examples and semantics

We mix functions and pattern-matching in a single entity.

- $x\colon A \vdash t\colon C$
- $y\colon B \vdash t'\colon C$
- $t \perp t'$

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & t \\ \mathtt{inj}_r\, y & \mapsto & t' \end{array} \right\} : A \oplus B \leftrightarrow C$$

You can also form: $\left\{ \begin{array}{lcl} t & \mapsto & \mathtt{inj}_l\, x \\ t' & \mapsto & \mathtt{inj}_r\, y \end{array} \right\} : C \leftrightarrow A \oplus B$

Functions are as general as possible: $\{t_1 \mapsto t'_1 \mid \cdots \mid t_m \mapsto t'_m\}$.

$$[\![\{t_1 \mapsto t'_1 \mid \cdots \mid t_m \mapsto t'_m\}]\!] = \sum_i [\![t'_i]\!]\, [\![t_i]\!]^\dagger$$

$$[\![A]\!] \xrightarrow{[\![t_i]\!]^\dagger} [\![\Delta_i]\!] \xrightarrow{[\![t'_i]\!]} [\![B]\!]$$

Classical symmetric pattern-matching

$$
\begin{array}{lllll}
\text{(Ground types)} & A, B & ::= & \mathtt{I} \mid A \oplus B \mid A \otimes B \mid \\
\text{(Function types)} & T_1, T_2 & ::= & A \leftrightarrow B \mid \\
 & & & \\
\text{(Unit term)} & t, t_1, t_2 & ::= & * \\
\text{(Pairing)} & & & \mid t_1 \otimes t_2 \\
\text{(Injections)} & & & \mid \mathtt{inj}_l\ t \mid \mathtt{inj}_r\ t \\
\text{(Function application)} & & & \mid \omega\ t \\
 & & & \\
\text{(Abstraction)} & \omega & ::= & \{t_1 \mapsto t_1' \mid \cdots \mid t_m \mapsto t_m'\}
\end{array}
$$

Classical symmetric pattern-matching

$$
\begin{array}{lllll}
\text{(Ground types)} & A, B & ::= & \mathtt{I} \mid A \oplus B \mid A \otimes B \mid X \mid \mu X.A \\
\text{(Function types)} & T_1, T_2 & ::= & A \leftrightarrow B \mid \\
 & & & \\
\text{(Unit term)} & t, t_1, t_2 & ::= & * \\
\text{(Pairing)} & & & \mid t_1 \otimes t_2 \\
\text{(Injections)} & & & \mid \mathtt{inj}_l\ t \mid \mathtt{inj}_r\ t \\
\text{(Function application)} & & & \mid \omega\ t \\
\text{(Inductive terms)} & & & \mid \mathtt{fold}\ t \\
 & & & \\
\text{(Abstraction)} & \omega & ::= & \{t_1 \mapsto t_1' \mid \cdots \mid t_m \mapsto t_m'\} \\
\text{(Fixed points)} & & & \mid f \mid \textbf{fix}\ f.\omega
\end{array}
$$

Classical symmetric pattern-matching

$$
\begin{array}{lllll}
\text{(Ground types)} & A, B & ::= & \mathtt{I} \mid A \oplus B \mid A \otimes B \mid X \mid \mu X.A \\
\text{(Function types)} & T_1, T_2 & ::= & A \leftrightarrow B \mid T_1 \to T_2 \\
 & & & \\
\text{(Unit term)} & t, t_1, t_2 & ::= & * \\
\text{(Pairing)} & & & \mid t_1 \otimes t_2 \\
\text{(Injections)} & & & \mid \mathtt{inj}_l\ t \mid \mathtt{inj}_r\ t \\
\text{(Function application)} & & & \mid \omega\ t \\
\text{(Inductive terms)} & & & \mid \mathtt{fold}\ t \\
 & & & \\
\text{(Abstraction)} & \omega & ::= & \{t_1 \mapsto t_1' \mid \cdots \mid t_m \mapsto t_m'\} \\
\text{(Fixed points)} & & & \mid f \mid \mathbf{fix}\ f.\omega \\
\text{(Higher functions)} & & & \mid \lambda f.\omega \mid \omega_2 \omega_1
\end{array}
$$

λ-calculus with fixed points thanks to **DCPO**-enrichment of **PInj**.

And now, guarded

Reversible Guarded Recursion

(Ground types)	A, B	$::=$	$\mathtt{I} \mid A \oplus B \mid A \otimes B \mid \blacktriangleright A \mid$
(Function types)	T_1, T_2	$::=$	$A \leftrightarrow B \mid \blacktriangleright T \mid T_1 \to T_2$
(Unit term)	t, t_1, t_2	$::=$	$*$
(Pairing)			$\mid t_1 \otimes t_2$
(Injections)			$\mid \mathtt{inj}_l\ t \mid \mathtt{inj}_r\ t$
(Function application)			$\mid \omega\ t$
(Later modality)			$\mid \mathtt{next}\ t$
(Inductive terms)			$\mid \mathtt{fold}\ t$
(Abstraction)	ω	$::=$	$\{t_1 \mapsto t_1' \mid \cdots \mid t_m \mapsto t_m'\}$
(Higher modality)			$\mid \mathtt{next}\ \omega$
(Fixed points)			$\mid f \mid \mathbf{fix}\ f.\omega$
(Higher functions)			$\mid \lambda f.\omega \mid \omega_2 \omega_1$

Reversible Guarded Recursion

(Ground types)	A, B	$::=$	$\mathtt{I} \mid A \oplus B \mid A \otimes B \mid \blacktriangleright A \mid X \mid \mu X.A$
(Function types)	T_1, T_2	$::=$	$A \leftrightarrow B \mid \blacktriangleright T \mid T_1 \to T_2$
(Unit term)	t, t_1, t_2	$::=$	$*$
(Pairing)			$\mid t_1 \otimes t_2$
(Injections)			$\mid \mathtt{inj}_l\ t \mid \mathtt{inj}_r\ t$
(Function application)			$\mid \omega\ t$
(Later modality)			$\mid \mathtt{next}\ t$
(Inductive terms)			$\mid \mathtt{fold}\ t$
(Abstraction)	ω	$::=$	$\{t_1 \mapsto t_1' \mid \cdots \mid t_m \mapsto t_m'\}$ *but!*
(Higher modality)			$\mid \mathtt{next}\ \omega$
(Fixed points)			$\mid f \mid \mathbf{fix}\ f.\omega$
(Higher functions)			$\mid \lambda f.\omega \mid \omega_2 \omega_1$

Daggerability in the syntax

The function:

$$\{x \mapsto \texttt{next}\ x\} : A \to \blacktriangleright A$$

would not have a daggerable semantics.

Daggerability in the syntax

The function:

$$\{x \mapsto \texttt{next}\ x\} \colon A \to \blacktriangleright A$$

would not have a daggerable semantics.

Both terms on left and right need to have the same depth, to fit the condition: $\nu_Y^{\mathbf{C}} \circ f \circ \left(\nu_X^{\mathbf{C}}\right)^{\dagger}$

Daggerability in the syntax

The function:

$$\{x \mapsto \texttt{next}\ x\} \colon A \to \blacktriangleright A$$

would not have a daggerable semantics.

Both terms on left and right need to have the same depth, to fit the condition: $\nu^{\mathbf{C}}_{Y} \circ f \circ \left(\nu^{\mathbf{C}}_{X}\right)^{\dagger}$

Example

Lists of type A obtained as $[A] = \mu X.1 \oplus (A \otimes \blacktriangleright X)$.

Daggerability in the syntax

The function:

$$\{x \mapsto \texttt{next}\ x\} \colon A \to \blacktriangleright A$$

would not have a daggerable semantics.

Both terms on left and right need to have the same depth, to fit the condition: $\nu^{\mathbf{C}}_{Y} \circ f \circ \left(\nu^{\mathbf{C}}_{X}\right)^{\dagger}$

Example

Lists of type A obtained as $[A] = \mu X.1 \oplus (A \otimes \blacktriangleright X)$.

The depth condition still allows for the map function.

$$\texttt{map}(\omega) = \textbf{fix}\ \mathrm{f}.\ \left\{ \begin{array}{ll} [\,] & \mapsto [\,] \\ h :: t & \mapsto (\omega\ h) :: (\ \mathrm{f}\ \ t) \end{array} \right\} : [A] \leftrightarrow [B]$$

Daggerability in the syntax

The function:

$$\{x \mapsto \texttt{next}\ x\} \colon A \to \blacktriangleright A$$

would not have a daggerable semantics.

Both terms on left and right need to have the same depth, to fit the condition: $\nu^{\mathbf{C}}_{Y} \circ f \circ \left(\nu^{\mathbf{C}}_{X}\right)^{\dagger}$

Example

Lists of type A obtained as $[A] = \mu X.1 \oplus (A \otimes \blacktriangleright X)$.

The depth condition still allows for the map function.

$$\texttt{map}(\omega) = \textbf{fix}\ \mathrm{f}.\ \left\{ \begin{array}{ll} [\,] & \mapsto [\,] \\ h :: t & \mapsto (\omega\ h) :: (\ \mathrm{f}\ \ t) \end{array} \right\} : [A] \leftrightarrow [B]$$

But no infinite loops allowed.

Conclusion

We can create some artificial recursion out of thin air.

Conclusion

We can create some artificial recursion out of thin air.

The situation is trickier for (pure) quantum computation;

Conclusion

We can create some artificial recursion out of thin air.

The situation is trickier for (pure) quantum computation;

with limited expressivity.

Conclusion

We can create some artificial recursion out of thin air.

The situation is trickier for (pure) quantum computation;

with limited expressivity.

Thank you!