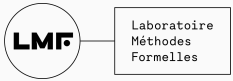
Semantics for a Turing-complete Reversible Programming Language with Inductive Types

Kostia CHARDONNET[1], Louis LEMONNIER[2] and Benoît VALIRON[3]

[1] University of Bologna ⟶ Inria Nancy

[2] LMF, Université Paris-Saclay ⟶ University of Edinburgh

[3] CentraleSupélec, LMF, Université Paris-Saclay



FSCD'24. 12th July 2024

Why study semantics?

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

Why study semantics?

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

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

Why study semantics?

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

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

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

Why study semantics?

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

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

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

Why study semantics?

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

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

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

Why study semantics?

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

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

$$\text{For example, } (\lambda x.x + 2)3 \to 3 + 2 \to 5.$$

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

Why study semantics?

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

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

$$\text{For example, } (\lambda x.x + 2)3 \to 3 + 2 \to 5.$$

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

Strong enough if relation between operational and denotational:

$$t \simeq t' \text{ iff } [\![t]\!] = [\![t']\!].$$

Why study semantics?

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

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

$$\text{For example, } (\lambda x.x + 2)3 \to 3 + 2 \to 5.$$

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

Strong enough if relation between operational and denotational:

$$t \simeq t' \text{ iff } [\![t]\!] = [\![t']\!].$$

- 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.

Reversible Programming

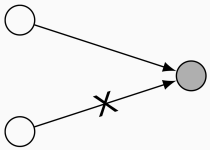
Originally

- Landauer and Bennett, 1961: Reversible Computation and Energy Dissipation.
- Programs are reversible: for a program $\mathtt{t}$, there is $\mathtt{t}^{-1}$ such that $\mathtt{t}; \mathtt{t}^{-1} = \mathtt{skip}$.
- Applications to quantum computing (we can chat about this later).

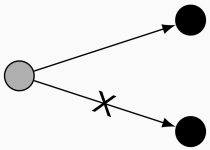
What we do

- Reversibility, but not totality.
- Improve the language through the model.
- Soundness and adequacy.

Backward determinism



Forward determinism



[Kaarsgaard&Rennela21]

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality

Partial inverse

Union of morphisms

Order

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality Restriction category: $f \mapsto \bar{f}$.

Partial inverse

Union of morphisms

Order

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality

Restriction category: $f \mapsto \bar{f}$.

$f \circ \bar{f} = f, \quad \bar{f} \circ \bar{g} = \bar{g} \circ \bar{f}, \quad \overline{f \circ \bar{g}} = \bar{f} \circ \bar{g}, \quad \bar{h} \circ f = f \circ \overline{h \circ f}.$

Partial inverse

Union of morphisms

Order

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality

Restriction category: $f \mapsto \overline{f}$.

$f \circ \overline{f} = f$, $\overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}$, $\overline{f \circ \overline{g}} = \overline{f} \circ \overline{g}$, $\overline{h} \circ f = f \circ \overline{h \circ f}$.

$h : \begin{smallmatrix} a \\ & \searrow \\ c & & \searrow \\ & & & b \end{smallmatrix}$ $\overline{h} : \begin{smallmatrix} a \\ & \searrow \\ c & & \rightarrow \\ & & & b \end{smallmatrix}$

Partial inverse

Union of morphisms

Order

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality — Restriction category: $f \mapsto \bar{f}$.

$f \circ \bar{f} = f$, $\bar{f} \circ \bar{g} = \bar{g} \circ \bar{f}$, $\overline{f \circ \bar{g}} = \bar{f} \circ \bar{g}$, $\bar{h} \circ f = f \circ \overline{h \circ f}$.

$h : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \searrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$ $\quad$ $\bar{h} : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \rightarrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$

Partial inverse — Inverse category: $f \mapsto f^\circ$. $f^\circ \circ f = \bar{f}$ and $f \circ f^\circ = \overline{f^\circ}$.

Union of morphisms

Order

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality

Restriction category: $f \mapsto \overline{f}$.

$f \circ \overline{f} = f$, $\overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}$, $\overline{f \circ \overline{g}} = \overline{f} \circ \overline{g}$, $\overline{h} \circ f = f \circ \overline{h \circ f}$.

$h : \begin{smallmatrix} a & & b \\ & \searrow & \searrow \\ & c & & b \end{smallmatrix}$ $\overline{h} : \begin{smallmatrix} a & \to & a \\ c & & b \end{smallmatrix}$

Partial inverse

Inverse category: $f \mapsto f^\circ$. $f^\circ \circ f = \overline{f}$ and $f \circ f^\circ = \overline{f^\circ}$.

$h^\circ : \begin{smallmatrix} & a & a \\ & \nearrow & \nearrow \\ c & & b \end{smallmatrix}$

Union of morphisms

Order

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality — Restriction category: $f \mapsto \overline{f}$.

$f \circ \overline{f} = f$, $\overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}$, $\overline{f \circ \overline{g}} = \overline{f} \circ \overline{g}$, $\overline{h} \circ f = f \circ \overline{h \circ f}$.

$h : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \searrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$ $\quad$ $\overline{h} : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \rightarrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$

Partial inverse — Inverse category: $f \mapsto f^\circ$. $f^\circ \circ f = \overline{f}$ and $f \circ f^\circ = \overline{f^\circ}$.

$h^\circ : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \nearrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$

Union of morphisms — Join: $\bigvee_{s \in S} s$.

Order

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality

Restriction category: $f \mapsto \overline{f}$.

$f \circ \overline{f} = f, \quad \overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}, \quad \overline{f \circ \overline{g}} = \overline{f} \circ \overline{g}, \quad \overline{h} \circ f = f \circ \overline{h \circ f}.$

$h : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \searrow \begin{smallmatrix} a \\ b \end{smallmatrix}$ $\qquad \overline{h} : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \rightarrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$

Partial inverse

Inverse category: $f \mapsto f^\circ$. $\quad f^\circ \circ f = \overline{f}$ and $f \circ f^\circ = \overline{f^\circ}$.

$h^\circ : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \nearrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$

Union of morphisms

Join: $\bigvee_{s \in S} s$.

$$f \circ \left(\bigvee_{s \in S} s \right) = \bigvee_{s \in S} fs, \quad \left(\bigvee_{s \in S} s \right) \circ g = \bigvee_{s \in S} sg.$$

Order

The categorical model: inverse categories

Category theory around reversibility is well studied [Axelsen&Kaarsgaard16].

Domain and partiality

Restriction category: $f \mapsto \overline{f}$.

$f \circ \overline{f} = f$, $\overline{f} \circ \overline{g} = \overline{g} \circ \overline{f}$, $\overline{f \circ \overline{g}} = \overline{f} \circ \overline{g}$, $\overline{h} \circ f = f \circ \overline{h \circ f}$.

$h : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \searrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$ $\quad$ $\overline{h} : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \to \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$

Partial inverse

Inverse category: $f \mapsto f^\circ$. $f^\circ \circ f = \overline{f}$ and $f \circ f^\circ = \overline{f^\circ}$.

$h^\circ : \begin{smallmatrix} a \\ b \\ c \end{smallmatrix} \nearrow \begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$

Union of morphisms

Join: $\bigvee_{s \in S} s$.

$$f \circ \left(\bigvee_{s \in S} s \right) = \bigvee_{s \in S} fs, \quad \left(\bigvee_{s \in S} s \right) \circ g = \bigvee_{s \in S} sg.$$

Order

$f \leq g$ if $g\overline{f} = f$

Gives a dcpo structure to homsets.

How do you extract syntax from inverse categories?

Cartesian closed category

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\to$.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\to$.
  - Type $A \to B$.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\to$.
  - Type $A \to B$.
  - Constructor $\lambda x.t$.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\to$.
  - Type $A \to B$.
  - Constructor $\lambda x.t$.

Inverse category

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\to$.
  - Type $A \to B$.
  - Constructor $\lambda x.t$.

Inverse category

- Not cartesian, but monoidal tensor.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\to$.
  - Type $A \to B$.
  - Constructor $\lambda x.t$.

Inverse category

- Not cartesian, but monoidal tensor.
  - Type $A \otimes B$.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\to$.
  - Type $A \to B$.
  - Constructor $\lambda x.t$.

Inverse category

- Not cartesian, but monoidal tensor.
  - Type $A \otimes B$.
  - Linear type system.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\rightarrow$.
  - Type $A \rightarrow B$.
  - Constructor $\lambda x.t$.

Inverse category

- Not cartesian, but monoidal tensor.
  - Type $A \otimes B$.
  - Linear type system.
- Not monoidal closed.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\rightarrow$.
  - Type $A \rightarrow B$.
  - Constructor $\lambda x.t$.

Inverse category

- Not cartesian, but monoidal tensor.
  - Type $A \otimes B$.
  - Linear type system.
- Not monoidal closed.
  - No ground function type.

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\rightarrow$.
  - Type $A \rightarrow B$.
  - Constructor $\lambda x.t$.

Inverse category

- Not cartesian, but monoidal tensor.
  - Type $A \otimes B$.
  - Linear type system.
- Not monoidal closed.
  - No ground function type.
  - Is there a way to form functions?

How do you extract syntax from inverse categories?

Cartesian closed category

- Cartesian product $\times$.
  - Type $A \times B$.
  - Constructor $\langle t_1, t_2 \rangle$.
- Right adjoint to the tensor $\rightarrow$.
  - Type $A \rightarrow B$.
  - Constructor $\lambda x.t$.

Inverse category

- Not cartesian, but monoidal tensor.
  - Type $A \otimes B$.
  - Linear type system.
- Not monoidal closed.
  - No ground function type.
  - Is there a way to form functions?

Hopefully, there is a way to cheat.

Our new functions

We can cheat with our partial inverse!

Our new functions

We can cheat with our partial inverse!

$$\begin{array}{lcl} [\![\Delta \vdash t\colon A]\!] & : & [\![\Delta]\!] \to [\![A]\!] \\ [\![\Delta \vdash t\colon A]\!]^{\circ} & : & [\![A]\!] \to [\![\Delta]\!] \end{array}$$

Our new functions

We can cheat with our partial inverse!

$$\begin{array}{lcl} [\![\Delta \vdash t\colon A]\!] & : & [\![\Delta]\!] \to [\![A]\!] \\ [\![\Delta \vdash t\colon A]\!]^{\circ} & : & [\![A]\!] \to [\![\Delta]\!] \end{array}$$

What do we do with this?

Our new functions

We can cheat with our partial inverse!

$$\begin{array}{lcl} [\![\Delta \vdash t\colon A]\!] & : & [\![\Delta]\!] \to [\![A]\!] \\ [\![\Delta \vdash t\colon A]\!]^{\circ} & : & [\![A]\!] \to [\![\Delta]\!] \end{array}$$

What do we do with this?

Given $\quad \Delta \vdash t\colon A \qquad \Delta \vdash t'\colon B$

We form a function $t \mapsto t'\colon A \leftrightarrow B$, Whose semantics is

Our new functions

We can cheat with our partial inverse!

$$\begin{array}{lcl} [\![\Delta \vdash t\colon A]\!] & : & [\![\Delta]\!] \to [\![A]\!] \\ [\![\Delta \vdash t\colon A]\!]^{\circ} & : & [\![A]\!] \to [\![\Delta]\!] \end{array}$$

What do we do with this?

Given $\quad \Delta \vdash t\colon A \qquad \Delta \vdash t'\colon B$

We form a function $t \mapsto t'\colon A \leftrightarrow B$, Whose semantics is

$$[\![A]\!] \overset{[\![\Delta \vdash t\,:\,A]\!]^{\circ}}{\longrightarrow} [\![\Delta]\!] \overset{[\![\Delta \vdash t'\,:\,B]\!]}{\longrightarrow} [\![B]\!]$$

Our new functions

We can cheat with our partial inverse!

$$\begin{array}{lcl} [\![\Delta \vdash t\colon A]\!] & : & [\![\Delta]\!] \to [\![A]\!] \\ [\![\Delta \vdash t\colon A]\!]^{\circ} & : & [\![A]\!] \to [\![\Delta]\!] \end{array}$$

What do we do with this?

Given $\quad \Delta \vdash t\colon A \qquad \Delta \vdash t'\colon B$

We form a function $t \mapsto t'\colon A \leftrightarrow B$, Whose semantics is

$$[\![A]\!] \xrightarrow{[\![\Delta \vdash t\,:\,A]\!]^{\circ}} [\![\Delta]\!] \xrightarrow{[\![\Delta \vdash t'\,:\,B]\!]} [\![B]\!]$$

This is a reversible function! We have $(t \mapsto t')^{-1} = t' \mapsto t$, whose semantics is:

Our new functions

We can cheat with our partial inverse!

$$\begin{array}{lcl} [\![\Delta \vdash t\colon A]\!] & : & [\![\Delta]\!] \to [\![A]\!] \\ [\![\Delta \vdash t\colon A]\!]^{\circ} & : & [\![A]\!] \to [\![\Delta]\!] \end{array}$$

What do we do with this?

Given $\quad \Delta \vdash t\colon A \qquad \Delta \vdash t'\colon B$

We form a function $t \mapsto t'\colon A \leftrightarrow B$, Whose semantics is

$$[\![A]\!] \xrightarrow{[\![\Delta \vdash t\,:\,A]\!]^{\circ}} [\![\Delta]\!] \xrightarrow{[\![\Delta \vdash t'\,:\,B]\!]} [\![B]\!]$$

This is a reversible function! We have $(t \mapsto t')^{-1} = t' \mapsto t$, whose semantics is:

$$[\![B]\!] \xrightarrow{[\![\Delta \vdash t'\,:\,B]\!]^{\circ}} [\![\Delta]\!] \xrightarrow{[\![\Delta \vdash t\,:\,A]\!]} [\![A]\!]$$

Together with pattern-matching

With a sum type $\oplus$:

$$\frac{\Delta \vdash t\colon A}{\Delta \vdash \mathtt{inj}_l\ t\colon A \oplus B} \qquad \frac{\Delta \vdash t\colon B}{\Delta \vdash \mathtt{inj}_r\ t\colon A \oplus B}$$

Together with pattern-matching

With a sum type $\oplus$:

$$\frac{\Delta \vdash t\colon A}{\Delta \vdash \mathtt{inj}_l\ t\colon A \oplus B} \qquad \frac{\Delta \vdash t\colon B}{\Delta \vdash \mathtt{inj}_r\ t\colon A \oplus B}$$

We introduce orthogonality:

Together with pattern-matching

With a sum type $\oplus$:

$$\frac{\Delta \vdash t\colon A}{\Delta \vdash \mathtt{inj}_l\ t\colon A \oplus B} \qquad \frac{\Delta \vdash t\colon B}{\Delta \vdash \mathtt{inj}_r\ t\colon A \oplus B}$$

We introduce orthogonality:

$$\frac{}{\mathtt{inj}_l\ t_1 \perp \mathtt{inj}_r\ t_2} \qquad \frac{t_1 \perp t_2}{C[t_1] \perp C[t_2]}$$

Together with pattern-matching

With a sum type $\oplus$:

$$\frac{\Delta \vdash t\colon A}{\Delta \vdash \mathtt{inj}_l\ t\colon A \oplus B} \qquad \frac{\Delta \vdash t\colon B}{\Delta \vdash \mathtt{inj}_r\ t\colon A \oplus B}$$

We introduce orthogonality:

$$\frac{}{\mathtt{inj}_l\ t_1 \perp \mathtt{inj}_r\ t_2} \qquad \frac{t_1 \perp t_2}{C[t_1] \perp C[t_2]}$$

Our functions are then:

$$\left\{ \begin{array}{ccc} t_1 & \mapsto & t_1' \\ t_2 & \mapsto & t_2' \\ & \vdots & \\ t_m & \mapsto & t_m' \end{array} \right\} : A \leftrightarrow B$$

whenever $\Delta_i \vdash t_i\colon A$ and $t_j \perp t_k$, $\quad \Delta_i \vdash t_i'\colon B$ and $t_j' \perp t_k'$.

Example and operational semantics

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, y & \mapsto & \mathtt{inj}_l\, y \end{array} \right\} : A \oplus B \leftrightarrow B \oplus A$$

Example and operational semantics

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, y & \mapsto & \mathtt{inj}_l\, y \end{array} \right\} : A \oplus B \leftrightarrow B \oplus A$$

Can be seen as:

- Partial morphisms joined together.
- With compatible domains.
- Whose inverse have compatible domains.

Operational semantics:

Example and operational semantics

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, y & \mapsto & \mathtt{inj}_l\, y \end{array} \right\} : A \oplus B \leftrightarrow B \oplus A$$

Can be seen as:

- Partial morphisms joined together.
- With compatible domains.
- Whose inverse have compatible domains.

Operational semantics:

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, x & \mapsto & \mathtt{inj}_l\, x \end{array} \right\} \mathtt{inj}_r\, v \rightarrow$$

Example and operational semantics

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, y & \mapsto & \mathtt{inj}_l\, y \end{array} \right\} : A \oplus B \leftrightarrow B \oplus A$$

Can be seen as:

- Partial morphisms joined together.
- With compatible domains.
- Whose inverse have compatible domains.

Operational semantics:

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, x & \mapsto & \mathtt{inj}_l\, x \end{array} \right\} \; \mathtt{inj}_r\, v \to (\mathtt{inj}_l\, x)[v/x] \to$$

Example and operational semantics

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, y & \mapsto & \mathtt{inj}_l\, y \end{array} \right\} : A \oplus B \leftrightarrow B \oplus A$$

Can be seen as:

- Partial morphisms joined together.
- With compatible domains.
- Whose inverse have compatible domains.

Operational semantics:

$$\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, x & \mapsto & \mathtt{inj}_l\, x \end{array} \right\} \; \mathtt{inj}_r\, v \to (\mathtt{inj}_l\, x)[v/x] \to \mathtt{inj}_l\, v$$

The mathematical recipe /ˈɹɛs.ɹ.pi/

Our category **C** such that:

The mathematical recipe /ˈɹɛs.ɪ.pi/

Our category **C** such that:

- Inverse category.
  - Partial inverse $(-)^\circ$.

The mathematical recipe /ˈɹɛs.ɹ.pi/

Our category **C** such that:

- Inverse category.
  - Partial inverse $(-)^{\circ}$.
  - Takes care of pattern-matching.

The mathematical recipe /ˈɪɛs.ɪ.pi/

Our category **C** such that:

- Inverse category.
  - Partial inverse $(-)^\circ$.
  - Takes care of pattern-matching.
- Rig structure.
  - Usual monoidal product $\otimes$.

The mathematical recipe /ˈɪɛs.ɪ.pi/

Our category **C** such that:

- Inverse category.
  - Partial inverse $(-)^\circ$.
  - Takes care of pattern-matching.
- Rig structure.
  - Usual monoidal product $\otimes$.
  - Disjointness tensor $\oplus$ with jointly monic injections.

The mathematical recipe /ˈɪɛs.ɪ.pi/

Our category **C** such that:

- Inverse category.
  - Partial inverse $(-)^\circ$.
  - Takes care of pattern-matching.
- Rig structure.
  - Usual monoidal product $\otimes$.
  - Disjointness tensor $\oplus$ with jointly monic injections.
- Join structure.
  - Compatible morphisms on their domain and codomain admit a join $\vee$.

The mathematical recipe /ˈɪɛs.ɪ.pi/

Our category **C** such that:

- Inverse category.
  - Partial inverse $(-)^\circ$.
  - Takes care of pattern-matching.
- Rig structure.
  - Usual monoidal product $\otimes$.
  - Disjointness tensor $\oplus$ with jointly monic injections.
- Join structure.
  - Compatible morphisms on their domain and codomain admit a join $\vee$.
  - Provides a nice structure on morphisms.

Example: Sets and partial injective functions **PInj**.

The mathematical recipe /ˈɪɛs.ɪ.pi/

Our category **C** such that:

- Inverse category.
  - Partial inverse $(-)^\circ$.
  - Takes care of pattern-matching.
- Rig structure.
  - Usual monoidal product $\otimes$.
  - Disjointness tensor $\oplus$ with jointly monic injections.
- Join structure.
  - Compatible morphisms on their domain and codomain admit a join $\vee$.
  - Provides a nice structure on morphisms.

Example: Sets and partial injective functions **PInj**.

$$\left[\!\!\left[\left\{ \begin{array}{lcl} \mathtt{inj}_l\, x & \mapsto & \mathtt{inj}_r\, x \\ \mathtt{inj}_r\, y & \mapsto & \mathtt{inj}_l\, y \end{array} \right\} : A \oplus B \leftrightarrow B \oplus A \right]\!\!\right] = [\![\mathtt{inj}_l\, x \mapsto \mathtt{inj}_r\, x]\!] \vee [\![\mathtt{inj}_r\, y \mapsto \mathtt{inj}_l\, y]\!]$$

How much can one express with this language?

To Infinity and Beyond

Some reading: [Axelsen&Kaarsgaard16] + [Fiore04] + some calculations.

$\longrightarrow$ Our category $\mathbf{C}$

To Infinity and Beyond

Some reading: [Axelsen&Kaarsgaard16] + [Fiore04] + some calculations.

$\longrightarrow$ Our category $\mathbf{C}$ is parameterised **DCPO**-algebraically ω-compact.

To Infinity and Beyond

Some reading: [Axelsen&Kaarsgaard16] + [Fiore04] + some calculations.

$\longrightarrow$ Our category $\mathbf{C}$ can model infinite data types $\mu X.A$.

To Infinity and Beyond

Some reading: [Axelsen&Kaarsgaard16] + [Fiore04] + some calculations.

$\longrightarrow$ Our category $\mathbf{C}$ can model infinite data types $\mu X.A$.

Examples:

$$\texttt{Nat} = \mu X.1 \oplus X$$

To Infinity and Beyond

Some reading: [Axelsen&Kaarsgaard16] + [Fiore04] + some calculations.

$\longrightarrow$ Our category $\mathbf{C}$ can model infinite data types $\mu X.A$.

Examples:

$$\texttt{Nat} = \mu X.1 \oplus X \qquad [A] = \mu X.1 \oplus (A \otimes X)$$

And we want to parse those infinite types:

To Infinity and Beyond

Some reading: [Axelsen&Kaarsgaard16] + [Fiore04] + some calculations.

$\longrightarrow$ Our category $\mathbf{C}$ can model infinite data types $\mu X.A$.

Examples:

$$\texttt{Nat} = \mu X.1 \oplus X \qquad [A] = \mu X.1 \oplus (A \otimes X)$$

And we want to parse those infinite types:

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

To Infinity and Beyond

Some reading: [Axelsen&Kaarsgaard16] + [Fiore04] + some calculations.

$\longrightarrow$ Our category $\mathbf{C}$ can model infinite data types $\mu X.A$.

Examples:

$$\texttt{Nat} = \mu X.1 \oplus X \qquad [A] = \mu X.1 \oplus (A \otimes X)$$

And we want to parse those infinite types:

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

Works in the category thanks to **DCPO**-enrichment.

To Infinity and Beyond

Some reading: [Axelsen&Kaarsgaard16] + [Fiore04] + some calculations.

$\longrightarrow$ Our category $\mathbf{C}$ can model infinite data types $\mu X.A$.

Examples:

$$\texttt{Nat} = \mu X.1 \oplus X \qquad [A] = \mu X.1 \oplus (A \otimes X)$$

And we want to parse those infinite types:

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

Works in the category thanks to **DCPO**-enrichment. $\boxed{\text{fix}(F) = \sup_n \{F^n(\bot)\}}$

Summary of the language (mandatory slide)

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

Summary of the language (mandatory slide)

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

Summary of the language (mandatory slide)

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

λ-calculus thanks to **DCPO**-enrichment.

The language is Turing complete! (even if it is reversible)

The language is Turing complete! (even if it is reversible)

⟵ ask this guy (Kostia Chardonnet, currently works in Nancy)

The language is Turing complete! (even if it is reversible)

⟵ ask this guy (Kostia Chardonnet, currently works in Nancy)

Roughly

- Reversible Turing Machines [Axelsen&Glück11].
  - Simulate your favourite Turing machines.
- Encode RTMs in our language:
  - Alphabet & states mapped to $\mathtt{I} \oplus \cdots \oplus \mathtt{I}$.
  - Tape as lists.
  - Functions simulating one-step transition of δ.
  - Iterate until final state.

A variety of diverging functions

Usual

$$\mathsf{fix}\ f^T\ .\ f^T : T$$

A variety of diverging functions

Usual

$$\mathsf{fix}\ f^{T}\ .\ f^{T} \colon T$$

Less usual

$$\mathsf{fix}\ f^{A \leftrightarrow B}\ .\ \{x \mapsto f\ x\} \colon A \leftrightarrow B$$

A variety of diverging functions

Usual

$$\mathsf{fix}\ f^{T}\ .\ f^{T} \colon T$$

Less usual

$$\mathsf{fix}\ f^{A \leftrightarrow B}\ .\ \{x \mapsto f\,x\} \colon A \leftrightarrow B$$

$$\begin{array}{rcl} (\mathsf{fix}\ f\ .\ \{x \mapsto f\,x\})\ v & \to & \{x \mapsto (\mathsf{fix}\ f\ .\ \{x \mapsto f\,x\})\ x\}\ v \\ & \to & ((\mathsf{fix}\ f\ .\ \{x \mapsto f\,x\})\ x)[v/x] \\ & = & (\mathsf{fix}\ f\ .\ \{x \mapsto f\,x\})\ v \end{array}$$

A variety of diverging functions

Usual

$$\text{fix } f^T \,.\, f^T \colon T$$

Less usual

$$\text{fix } f^{A \leftrightarrow B} \,.\, \{x \mapsto f\, x\} \colon A \leftrightarrow B$$

$$\begin{array}{rcl} (\text{fix } f \,.\, \{x \mapsto f\, x\})\ v & \to & \{x \mapsto (\text{fix } f \,.\, \{x \mapsto f\, x\})\ x\}\ v \\ & \to & ((\text{fix } f \,.\, \{x \mapsto f\, x\})\ x)[v/x] \\ & = & (\text{fix } f \,.\, \{x \mapsto f\, x\})\ v \end{array}$$

New challenger

A variety of diverging functions

Usual

$$\mathsf{fix}\ f^{T}\ .\ f^{T} \colon T$$

Less usual

$$\mathsf{fix}\ f^{A \leftrightarrow B}\ .\ \{x \mapsto f\,x\} \colon A \leftrightarrow B$$

$$\begin{array}{lcl} (\mathsf{fix}\ f\ .\ \{x \mapsto f\,x\})\ v & \to & \{x \mapsto (\mathsf{fix}\ f\ .\ \{x \mapsto f\,x\})\ x\}\ v \\ & \to & ((\mathsf{fix}\ f\ .\ \{x \mapsto f\,x\})\ x)[v/x] \\ & = & (\mathsf{fix}\ f\ .\ \{x \mapsto f\,x\})\ v \end{array}$$

New challenger

$$\lambda g^{A_1 \leftrightarrow B_1}\ .\ \mathsf{fix}\ f^{A_2 \leftrightarrow B_2}\ .\ \left\{ \begin{array}{lcl} \mathtt{inj}_l\,x & \mapsto & \mathtt{inj}_l\,(g\,x) \\ \mathtt{inj}_r\,y & \mapsto & \mathtt{inj}_r\,(f\,y) \end{array} \right\} \colon A_1 \oplus A_2 \leftrightarrow B_1 \oplus B_2$$

Let's conclude

- Typed and reversible programming language.

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!
- Soundness. if $t \to t'$, then $[\![t]\!] = [\![t']\!]$.

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!
- Soundness. if $t \to t'$, then $[\![t]\!] = [\![t']\!]$.
- Adequacy. if $[\![t]\!] \neq \bot$, then $t\downarrow$.

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!
- Soundness. if $t \to t'$, then $[\![t]\!] = [\![t']\!]$.
- Adequacy. if $[\![t]\!] \neq \bot$, then $t\downarrow$.
- (Full) completeness. Any computable partial injection is representable.

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!
- Soundness. if $t \to t'$, then $[\![t]\!] = [\![t']\!]$.
- Adequacy. if $[\![t]\!] \neq \bot$, then $t\downarrow$.
- (Full) completeness. Any computable partial injection is representable.

Take home message: no cartesian closure needed to have

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!
- Soundness. if $t \to t'$, then $[\![t]\!] = [\![t']\!]$.
- Adequacy. if $[\![t]\!] \neq \bot$, then $t\downarrow$.
- (Full) completeness. Any computable partial injection is representable.

Take home message: no cartesian closure needed to have functions,

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!
- Soundness. if $t \to t'$, then $[\![t]\!] = [\![t']\!]$.
- Adequacy. if $[\![t]\!] \neq \bot$, then $t\downarrow$.
- (Full) completeness. Any computable partial injection is representable.

Take home message: no cartesian closure needed to have functions, inductive types,

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!
- Soundness. if $t \to t'$, then $[\![t]\!] = [\![t']\!]$.
- Adequacy. if $[\![t]\!] \neq \bot$, then $t\downarrow$.
- (Full) completeness. Any computable partial injection is representable.

Take home message: no cartesian closure needed to have functions, inductive types, recursion.

Let's conclude

- Typed and reversible programming language.
- Categorical model: join inverse rig **DCPO**-categories.
- Turing completeness!
- Soundness. if $t \to t'$, then $[\![t]\!] = [\![t']\!]$.
- Adequacy. if $[\![t]\!] \neq \bot$, then $t\downarrow$.
- (Full) completeness. Any computable partial injection is representable.

Take home message: no cartesian closure needed to have functions, inductive types, recursion.

Self promotion:

- Kostia's PhD thesis, `https://theses.hal.science/tel-03959403v1`
- My PhD thesis, `https://theses.hal.science/tel-04625771v1`
- Benoît's habilitation, to be defended on 24th September 2024.