Plan

What do we need for recursion in CT
From syntactic to semantic
Subrecursion
Recursive Function Classes in Cartesian Categories

Joaquín Díaz Boïls

Universitat de València

MAP 2010, Logroño, November
Plan

What do we need for recursion in CT
From syntactic to semantic
Subrecursion

Joaquín Díaz Boîls
Recursive Function Classes in Cartesian Categories
What do we need for recursion in CT
What do we need for recursion in CT

- natural numbers object diagrams
- representation
- various Cartesian Categories
What do we need for recursion in CT

- natural numbers object diagrams
- representation
- various Cartesian Categories

From syntactic to semantic
Plan

What do we need for recursion in CT
- natural numbers object diagrams
- representation
- various Cartesian Categories

From syntactic to semantic
- free and syntactical structures
- categories of Algorithms
What do we need for recursion in CT
  - natural numbers object diagrams
  - representation
  - various Cartesian Categories

From syntactic to semantic
  - free and syntactical structures
  - categories of Algorithms

Subrecursion
Plan

What do we need for recursion in CT

- natural numbers object diagrams
- representation
- various Cartesian Categories

From syntactic to semantic

- free and syntactical structures
- categories of Algorithms

Subrecursion

- translations to CT
- Polarized Categories
In a Cartesian Category (cc) we have composition and a product.
The nno diagrams

In a Cartesian Category \((cc)\) we have composition and a product.

Recursion operator is obtained by means of an \(nno\).
In a Cartesian Category (cc) we have composition and a product.

Recursion operator is obtained by means of an nno.

**Definition**

A nno in a category \( C \) with terminal object 1 is \((N, z, s)\) with a commutative diagram.
In a Cartesian Category (cc) we have composition and a product.

Recursion operator is obtained by means of an nno.

**Definition**

A *nno* in a category $C$ with terminal object 1 is $(N, z, s)$ with a commutative diagram:

$$
\begin{array}{ccc}
1 & \xrightarrow{z} & N & \xrightarrow{s} & N \\
& \downarrow{m} & & \downarrow{m} & \\
& & A & \xrightarrow{g} & A \\
1 & \xrightarrow{f} & A & & \\
\end{array}
$$

where $m$ is unique.
Basic recursive structure is a *parametrized nno*. 
Basic recursive structure is a *parametrized nno*

**Definition**

A *parametrized nno* (pnno) is an *nno* \((N, z, s)\) for which there exists a commutative diagram.
Basic recursive structure is a *parametrized nno*

**Definition**

A *parametrized nno* (pnno) is an nno \((N, z, s)\) for which there exists a commutative diagram

\[
\begin{array}{ccc}
X & \xrightarrow{z_X} & NX \\
\downarrow & & \downarrow \ m \\
X & \xrightarrow{f} & A \\
\end{array}
\quad
\begin{array}{ccc}
NX & \xrightarrow{s_X} & NX \\
\downarrow \ m & & \downarrow m \\
A & \xrightarrow{g} & A \\
\end{array}
\]
Basic recursive structure is a \textit{parametrized nno}

\textbf{Definition}

A \textit{parametrized nno} (pnno) is an \textit{nno} \((N, z, s)\) for which there exists a commutative diagram

\[
\begin{array}{ccc}
X & \xrightarrow{z_X} & NX \\
\downarrow & & \downarrow m \\
X & \xrightarrow{f} & A \\
\end{array}
\quad \begin{array}{ccc}
NX & \xrightarrow{s_X} & NX \\
\downarrow m & & \downarrow m \\
A & \xrightarrow{g} & A \\
\end{array}
\]

We can speak of a \textit{weak nno} (wnno) if uniqueness is not required.
These definitions can be generalized
These definitions can be generalized

**Definition**

A *left nno* (*Inno*) in a *Monoidal Category* \( \mathcal{V} = (\mathcal{V}, \otimes, I, \alpha, \lambda, \varrho) \) is \((N, z, s)\) such that
These definitions can be generalized

**Definition**

A left nno (lnno) in a Monoidal Category $\mathcal{V} = (\mathcal{V}, \otimes, I, \alpha, \lambda, \rho)$ is $(N, z, s)$ such that

\[
\begin{array}{cccc}
I \otimes A & \xrightarrow{z \otimes A} & N \otimes A & \xrightarrow{s \otimes A} & N \otimes A \\
\downarrow{\lambda} & & \downarrow{m} & & \downarrow{m} \\
A & \xrightarrow{f} & B & \xrightarrow{g} & B
\end{array}
\]

commutes.
These definitions can be generalized

**Definition**

**A left nno (Inno) in a Monoidal Category** \( \mathcal{V} = (\mathcal{V}, \otimes, I, \alpha, \lambda, \rho) \) is \((N, z, s)\) such that

\[
\begin{array}{ccc}
I \otimes A & \xrightarrow{z \otimes A} & N \otimes A \\
\downarrow \lambda & & \downarrow m \\
A & \xrightarrow{f} & B \\
\end{array}
\quad
\begin{array}{ccc}
N \otimes A & \xrightarrow{s \otimes A} & N \otimes A \\
\downarrow m & & \downarrow m \\
N \otimes A & \xrightarrow{g} & B \\
\end{array}
\]

commutes
We define recursive function classes in $cc + nno$ by its representation
We define recursive function classes in $cc + nno$ by its representation.

**Definition**

We say that $f : \mathbb{N}^k \to \mathbb{N}$ is *representable* in a $cc + nno$ $C$ if there exists $\overline{f} : N^k \to N$ in $C$ such that

$$\overline{f} \langle \#n_1, \ldots, \#n_k \rangle = \#f(n_1, \ldots, n_k)$$
Given categorical structures we obtain recursive function classes:
Given categorical structures we obtain recursive function classes:

For total function classes in the form $f : \mathbb{N}^k \rightarrow \mathbb{N}$ we have:
Given categorical structures we obtain recursive function classes:

For total function classes in the form $f : \mathbb{N}^k \rightarrow \mathbb{N}$ we have:

1. $\mathcal{PR} = \{\text{Representables in } cc+wpnno\}$

2. $\mathcal{PR} \subseteq \{\text{Representables in } Topos+nno\} \subseteq TotalRec$
For partial functions we have:

\[\text{PartialRec} \subseteq \{\text{Representables in ccc+\text{nno}+\text{equalizers}}\}\]

And for a different kind of representation:

\[\text{PartialRec} = \{\text{Numeralwise Representables in ccc+\text{wnno}+\text{equalizers}}\}\]
For partial functions we have:

\[ \text{PartialRec} \subseteq \{ \text{Representables in } cc+nno+equalizers \} \]
For partial functions we have:

$$\text{PartialRec} \subseteq \{\text{Representables in } cc+nno+\text{equalizers}\}$$

And for a different kind of representation:
The results

For partial functions we have:

\[ \text{PartialRec} \subseteq \{ \text{Representables in } cc+nno+\text{equalizers} \} \]

And for a different kind of representation:

\[ \text{PartialRec} = \{ \text{Numeralwise Representables in } cc+wnno+\text{equalizers} \} \]
Free and syntactical structures

Two constructions
Two constructions

1. over $Grph$
Two constructions

1. over \textit{Grph}

**Peano-Lawvere Axiom in a category**

A category satisfies PL Axiom (or it is PL) if every object has an \textit{nno}.
Free and syntactical structures

Two constructions

1. over $Grph$

Peano-Lawvere Axiom in a category

A category satisfies PL Axiom (or it is PL) if every object has an

$\_$

PL categories are not in general cartesian nor are they endowed with a
terminal object
Free and syntactical structures

Two constructions

1. over $Grph$

Peano-Lawvere Axiom in a category

A category satisfies PL Axiom (or it is PL) if every object has an nno

$PL$ categories are not in general cartesian nor are they endowed with a terminal object

However $P = F_{PL}()$ has both
Let be $P = F_{PL}(\cdot)$ the Category of $PR$-formal functions
Let be \( P = F_{PL}(\cdot) \) the **Category of \( PR\)-formal functions**

**Definition**

---

Joaquín Díaz Boils

Recursive Function Classes in Cartesian Categories
Let be $P = F_{PL}(\cdot)$ the Category of $\mathcal{PR}$-formal functions

**Definition**

We define $P'$ as the $PL$ precategory generated $(\cdot)$.
Let be $P = F_{PL}(\cdot)$ the Category of $\mathcal{PR}$-formal functions

**Definition**

We define $P'$ as the $PL$ precategory generated $(\cdot)$

it will be called *precategory of $\mathcal{PR}$-programs*
Lema

*We call* $\mathcal{P}$ *the image graph* $\mathcal{P} \rightarrow \text{Set}$
We call $\mathbb{P}$ the image graph $P \rightarrow Set$

- its objects are $\mathbb{N}^p$
Plan
What do we need for recursion in CT
From syntactic to semantic
Subrecursion

Free and syntactical structures

Free and syntactical structures

Lema

*We call* \( \mathbb{P} \) *the image graph* \( P \rightarrow \text{Set} \)

- *its objects are* \( \mathbb{N}^p \)
- *its morphisms* \( f : \mathbb{N}^p \rightarrow \mathbb{N}^q \) *such that* \( f = (f_0, f_1, ..., f_{q-1}) \)
  *where* \( f_i \in \mathcal{PR} \)

Joaquin Diaz Boils
Recursive Function Classes in Cartesian Categories
Lema

We call $\mathbb{P}$ the image graph $P \rightarrow \text{Set}$

- its objects are $\mathbb{N}^p$
- its morphisms $f : \mathbb{N}^p \rightarrow \mathbb{N}^q$ such that $f = (f_0, f_1, \ldots, f_{q-1})$ where $f_i \in \mathcal{PR}$

$\mathbb{P}$ can (only?) be characterized by equivalence relations in $P$
Free and syntactical structures

We can summarize as
We can summarize as

\[
\begin{array}{c}
(\cdot) \\
\downarrow \\
\downarrow \\
\uparrow \\
\uparrow \\
\downarrow \\
\downarrow \\
\downarrow \\

P' \\
P \\
P' \\
P
\end{array}
\]

\(P'\) is a syntactical construction while \(P\) is a category with semantics.
Free and syntactical structures

We can summarize as

\[ (\cdot) \rightarrow P' \rightarrow P \rightarrow \mathbb{P} \rightarrow (\cdot) \]

\( P' \) is a syntactical construction while \( \mathbb{P} \) is a category with semantics
Categories of Algorithms

2. over $\text{Set}$
Categories of Algorithms

2. over $\textbf{Set}$

Concept of algorithm is bounded by
2. over $\text{Set}$

Concept of algorithm is bounded by
Categories of Algorithms

2. over Set

Concept of algorithm is bounded by

1. the implementations that we handle: programs
2. over Set

Concept of algorithm is bounded by

1. the implementations that we handle: programs
2. formalizations that are known: recursive functions
There is a tree for any $PR$-program labelling the edges with...
There is a tree for any $PR$-program labelling the edges with

- $C$ (composition),
- $R$ (recursion),
- $B$ (bracket)
There is a tree for any \( PR \)-program labelling the edges with

\[
\text{C (composition), R (recursion) and B (bracket)}
\]
There is a tree for any $PR$-program labelling the edges with $C$ (composition), $R$ (recursion) and $B$ (bracket).

- nodes are $\mathbb{N}^k$
There is a tree for any \( \mathcal{PR} \)-program labelling the edges with

\[ C \text{ (composition), } R \text{ (recursion) and } B \text{ (bracket)} \]

- nodes are \( \mathbb{N}^k \)
- edges are \( \mathcal{PR} \)-functions generated from \( z, s \) and \( \pi_i^k \)
Categories of Algorithms

There is a tree for any $PR$-program labelling the edges with

\[ C \text{ (composition)}, \ R \text{ (recursion)} \text{ and } B \text{ (bracket)} \]

- nodes are $\mathbb{N}^k$
- edges are $PR$-functions generated from $z$, $s$ and $\pi^k_i$

Definition

We call this graph $PR_{desc}$
There is a tree for any $PR$-program labelling the edges with

$C$ (composition), $R$ (recursion) and $B$ (bracket)

- nodes are $\mathbb{N}^k$
- edges are $PR$-functions generated from $z$, $s$ and $\pi^k_i$

Definition
We call this graph $PR\text{desc}$
in it has all descriptions about how to compute all $PR$-functions
Categories of Algorithms

Essentially equal $\sim$ in $\mathcal{PR}_{\text{desc}}$ gives equivalence classes
Essentially equal $\sim$ in $\mathcal{PR}_{\text{desc}}$ gives equivalence classes

We construct by prunning
Essentially equal $\sim$ in $\mathcal{PR}_{\text{desc}}$ gives equivalence classes

We construct by *prunning*

$$\frac{\mathcal{PR}_{\text{desc}}}{\sim} = \mathcal{PR}_{\text{Alg}}$$

as the free initial category $F_{\text{CatxN}}(\emptyset)$ in $\text{CatxN}$
Categories of Algorithms

Reducing by \( \approx \) (run the same operation) we have
Reducing by $\approx$ (run the same operation) we have

$$\frac{\mathcal{PRA}_{\text{Alg}}}{\approx} = \mathcal{PRA}_{\text{Func}}$$
Reducing by $\approx$ (run the same operation) we have

$$\mathcal{PRA}_{\text{Alg}} \approx = \mathcal{PRA}_{\text{Func}}$$

and the schema
Reducing by \( \approx \) (run the same operation) we have

\[
\frac{\mathcal{PRAlg}}{\approx} = \mathcal{PRFunc}
\]

and the schema

\[
\begin{array}{c}
\mathcal{PRdesc} \\
\mathcal{PRFunc}
\end{array} 
\xymatrix{ 
\emptyset & \mathcal{PRAlg} \\
& \mathcal{PRdesc} \\
& \mathcal{PRFunc}
\}
\]
Free and syntactical structures

Similar constructions
Similar constructions

- initial PRU $\mathcal{E}$ (Pfender)
Similar constructions

- initial PRU $\mathcal{E}$ (Pfender)

- Freyd Cover $\mathcal{FC}$ from a cc $\mathcal{C} + nno$ (Román)
Free and syntactical structures

Similar constructions

- initial PRU $\mathcal{E}$ (Pfender)
- Freyd Cover $\mathcal{FC}$ from a cc $\mathcal{C} + nno$ (Román)
- free Monoidal Category + Inno $\Phi(\emptyset)$ (Román-Paré)
Transitions to CT

Classes in *Gzegorzycyk Hierarchy* can be defined by *bounding arithmetics*
Classes in *Gzegorzcyk Hierarchy* can be defined by *bounding arithmetics*.

Consider the smallest derivations set.

1. Containing a derivation of every initial function $0$,
2. $S(x) = x + 1$,
3. $P(x) = \max(0, x - 1)$
4. and conditional $C(x, y, z) =
   \begin{cases} 
   y & \text{if } x = 0 \\
   z & \text{else}
   \end{cases}$

Closed under the derivation rules:
1. **Full composition**: given derivations $h$ and $g_1, \ldots, g_m$ we derive $f(x) = h(g_1(x_1), \ldots, g_m(x_m))$
2. **Full primitive recursiveness**: for $g$ and $h$ if $x \neq 0$ we derive $f(x, y) = h(x, y, f(Px, y))$
Classes in Gzegorzycyk Hierarchy can be defined by bounding arithmetics

Consider the smallest derivations set

- containing a derivation of every initial function 0, \( Sx = x + 1 \),
- \( Px = \text{max}(0, x - 1) \) and conditional \( C(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases} \)
Classes in *Gzegorzycyk Hierarchy* can be defined by *bounding arithmetics*

Consider the smallest derivations set

1. containing a derivation of every initial function $0$, $Sx = x + 1$, $Px = \max(0, x - 1)$ and *conditional* $C(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases}$

2. closed under the derivation rules
Classes in *Gzregorczyk Hierarchy* can be defined by *bounding arithmetics*

Consider the smallest derivations set

1. containing a derivation of every initial function $0$, $Sx = x + 1$, $Px = \max(0, x - 1)$ and *conditional* $C(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases}$

2. closed under the derivation rules

   1. *full composition*: given derivations $h$ and $g_1, ..., g_m$ we derive

      $$f(\bar{x}) = h(g_1(\bar{x}^1), ..., g_m(\bar{x}^m))$$
Classes in *Gzregorzyk Hierarchy* can be defined by *bounding arithmetics*

Consider the smallest derivations set

1. containing a derivation of every initial function \( 0, Sx = x + 1, \)
   \[ Px = \max(0, x - 1) \]
   and *conditional* \( C(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases} \)

2. closed under the derivation rules

   1. **full composition**: given derivations \( h \) and \( g_1, ..., g_m \) we derive
      \[ f(x) = h(g_1(x^1), ..., g_m(x^m)) \]
   
   2. **full primitive recursiveness**: for \( g \) and \( h \) if \( x \neq 0 \) we derive
      \[ f(x, y) = h(x, y, f(Px, y)) \]
Complexity lower than $PR$ can be modeled using *ramified recursion*
Complexity lower than $\mathcal{PR}$ can be modeled using *ramified recursion*

it is based in *comprehension* or
Complexity lower than \( PR \) can be modeled using \textit{ramified recursion}

it is based in \textit{comprehension} or

\begin{itemize}
\item Axiom schema of specification
\end{itemize}
Complexity lower than $PR$ can be modeled using *ramified recursion*

it is based in *comprehension* or

**Axiom schema of specification**

*Any definable subclass of a set is a set*
Complexity lower than $\mathcal{PR}$ can be modeled using *ramified recursion*. It is based in *comprehension* or

**Axiom schema of specification**

*Any definable subclass of a set is a set*

Instead of *bounding induction* we use weaker subsystems.
Complexity lower than $\mathcal{PR}$ can be modeled using \textit{ramified recursion}

it is based in \textit{comprehension} or

\textbf{Axiom schema of specification}

\textit{Any definable subclass of a set is a set}

Instead of \textit{bounding induction} we use weaker subsystems

\textbf{We use two kinds of arguments: normal and safe}
Plan
What do we need for recursion in CT
From syntactic to semantic
Subrecursion

Translations to CT
Polarized Categories

Translations to CT

Definition

Joaquín Díaz Boils
Recursive Function Classes in Cartesian Categories
Definition

Subsets $\mathbb{N}_{k+1}$ that *make recursion* having $\mathbb{N}_0, \ldots, \mathbb{N}_k$ are called *tiers*.
Definition

Subsets $\mathbb{N}_{k+1}$ that make recursion having $\mathbb{N}_{o}, ..., \mathbb{N}_{k}$ are called tiers

We calculate tier of a derivation $f \in \mathcal{PR}$ as
Definition
Subsets $\mathbb{N}_{k+1}$ that make recursion having $\mathbb{N}_0, \ldots, \mathbb{N}_k$ are called tiers.

We calculate tier of a derivation $f \in PR$ as

- $\rho(f) = 0$ if $f$ is an initial function.
Plan

What do we need for recursion in CT
From syntactic to semantic
Subrecursion

Translations to CT
Polarized Categories

Translations to CT

Definition

Subsets $\mathbb{N}_{k+1}$ that make recursion having $\mathbb{N}_0, ..., \mathbb{N}_k$ are called tiers

We calculate tier of a derivation $f \in PR$ as

- $\rho(f) = 0$ if $f$ is an initial function

- $\rho(f) = \max\{\rho(h), \rho(g_1), ..., \rho(g_m)\}$ if $f$ is defined by full composition of derivations $h$ and $g_1, ..., g_m$
Transitions to CT

Definition
Subsets $\mathbb{N}_{k+1}$ that make recursion having $\mathbb{N}_0, \ldots, \mathbb{N}_k$ are called tiers.

We calculate tier of a derivation $f \in PR$ as

- $\rho(f) = 0$ if $f$ is an initial function
- $\rho(f) = \max\{\rho(h), \rho(g_1), \ldots, \rho(g_m)\}$ if $f$ is defined by full composition of derivations $h$ and $g_1, \ldots, g_m$
- $\rho(f) = \max\{\rho(g), 1 + \rho(h)\}$ if $f$ is defined by full primitive recursion of derivations $g$ and $h$
$\varepsilon_1$, $\varepsilon_2$ and $\varepsilon_3$ of Gzegorzcyk have been constructed from...
\( \varepsilon_1, \varepsilon_2 \) and \( \varepsilon_3 \) of Gzegorzcyk have been constructed from

*Doctrines with*
$\varepsilon_1$, $\varepsilon_2$ and $\varepsilon_3$ of Gzegorzcyk have been constructed from

**Doctrines with**

1. an *SM 2-Comprehension*
Translating to CT

\( \varepsilon_1, \varepsilon_2 \) and \( \varepsilon_3 \) of Gzegorzcyk have been constructed from

\begin{itemize}
  \item \textit{1} an \textit{SM 2-Comprehension}
  \item \textit{2} two \textit{tiers} \( N_0, N_1 \) of numerals with \textit{dyadics}
\end{itemize}

\[ I \xrightarrow{z} N_k \xrightarrow{s_k} N_k \]
Translating to CT

$\varepsilon_1$, $\varepsilon_2$ and $\varepsilon_3$ of Gzegorzcyk have been constructed from

**Doctrines with**

1. an *SM 2-Comprehension*
2. two *tiers* $N_0$, $N_1$ of numerals with *dyadics*

$$I \xrightarrow{z} N_k \xrightarrow{s_k} N_k$$

3. *ramified recursion*
Theorems with

1. an SM 2-Comprehension
2. two tiers $N_0$, $N_1$ of numerals with dyadics

$$I \xrightarrow{z} N_k \xrightarrow{s_k} N_k$$

3. ramified recursion
Polarized Categories

Polarization is a way to see ramification in CT
Polarized Categories

**Polarization** is a way to see ramification in CT

---

**Definition**

A *Polarized Strong Category* (SPolCat) consists of
**Polarized Categories**

**Definition**

A *Polarized Strong Category* (SPolCat) consists of

- a module $M : C \times D \to D$ where
Polarization is a way to see ramification in CT

Definition

A Polarized Strong Category (SPolCat) consists of

- a module $M : C \times D \rightarrow D$ where

$C$ is a cc (the opponent) and $D$ a category (the player)
Polarization is a way to see ramification in CT

Definition

A Polarized Strong Category (SPolCat) consists of

- a module $M : C \times D \to D$ where

  $C$ is a cc (the opponent) and $D$ a category (the player)

- endowed with a strong composition

$$
(C_1, D_1) \xrightarrow{f} D_2 \quad (C_2, D_2) \xrightarrow{g} D_3
$$

$$
(C_1 \times C_2, D_1) \xrightarrow{f \cdot g} D_3
$$
Plan
What do we need for recursion in CT
From syntactic to semantic
Subrecursion

Translations to CT
Polarized Categories

Polarized Categories

Definition
Polarized Categories

Definition

A *Polarized Functor* of a *SPolCat* $C$ consists of

A Polarized Functor of a $SPolCat$ $C$ consists of
Polarized Categories

Definition

A Polarized Functor of a SPolCat $C$ consists of

- functors $F_p : \mathcal{D} \to \mathcal{D}$ and $F_o : \mathcal{C} \to \mathcal{C}$
Definition

A *Polarized Functor* of a $SPolCat$ $C$ consists of:

- functors $F_p : D \to D$ and $F_o : C \to C$
- with $F_{op}$ acting
Polarized Categories

Definition

A Polarized Functor of a SPolCat $\mathcal{C}$ consists of

- functors $F_p : \mathcal{D} \rightarrow \mathcal{D}$ and $F_o : \mathcal{C} \rightarrow \mathcal{C}$

- with $F_{op}$ acting

$$(C, 1) \xrightarrow{f} D$$

$$(F_o(C), 1) \xrightarrow{F_{op}(f)} F_p(D)$$

Example $\otimes$ for $\mathcal{D}$ and $\times$ for $\mathcal{C}$ form a polarized functor.
Polarized Categories

**Definition**

A *Polarized Functor* of a *SPolCat* $C$ consists of

- functors $F_p : D \rightarrow D$ and $F_o : C \rightarrow C$

- with $F_{op}$ acting

\[
\begin{align*}
(C, 1) &\xrightarrow{f} D \\
(F_o(C), 1) &\xrightarrow{F_{op}(f)} F_p(D)
\end{align*}
\]

**Example**

⊗ for $D$ and × for $C$ form a polarized functor

Joaquín Díaz Boïls

Recursive Function Classes in Cartesian Categories
Polarized Categories

We can do *comprehended recursion* on fixed points over $F_{op}$.
We can do *comprehended recursion* on fixed points over $F_{op}$

Let $F^*$ be a free algebra generated by
We can do comprehended recursion on fixed points over $F_{op}$

Let $F^*$ be a free algebra generated by
- contexts $C = 1$ and $D = Nat$
We can do comprehended recursion on fixed points over $F_{op}$.

Let $F^*$ be a free algebra generated by

- contexts $C = 1$ and $D = Nat$
- constructors

$Zero : 1 \rightarrow Nat$ and $Succ : Nat \rightarrow Nat$
$F^*$ is a fixed point for a polarized functor in the form

\[
F_k(Z)
\]
$F^*$ is a fixed point for a polarized functor in the form

$$\sum_k F_k(Z)$$
$F^*$ is a fixed point for a polarized functor in the form

$$\sum_k F_k(Z)$$

Can we construct a $SPolCat$ from this to get a general form of ramified recursion in CT?